```
                rc = false;
        }
        else    {
                //      set the threshold for the group objects, since they apply
                //      the threshold
                groupEditDistThresh = aThresh;
                for (int i = 0; i < TDS_NUM_CULTURES; i++)
                        groupData[i]->setEditDistThresh(groupEditDistThresh);
                simpleEditDistancizer->setThreshold(groupEditDistThresh);
        }

        return rc;
}


/*

void    CTDSDlg::calcShouldMatchOnUCV(vector<CString> &queryUCVs)
{
                                                                        numVariantsForWatchName;
        int                                                             numVariantsForQueryName;
        int
        vector<CString>         *watchUCVs = NULL;
        int                                                             i, j;
        char                                                    spacedWatchName[TDS_MAX_NAME + 3];
        const char *                    queryUCV;
        const char *                    watchUCV;
        tds_vargen_code         genVarRC;



        numVariantsForQueryName = queryUCVs.size();
        //      get the variants for the watch name.
        sprintf(spacedWatchName, " %s ", watchName);
        strupr(spacedWatchName);
        watchUCVs = ruleSet.genVariants(spacedWatchName,

=>                                      TRUE,           //

=>                                      TRUE,           //

=>                                      UCV_LEN_IN_MAP,

=>                                      genVarRC);
        numVariantsForWatchName = watchUCVs->size();

        //      now we have the UCVs for both the query and the watch name.
        //      so go through each pairing and see if we have a match.  If
        //      we do, save the UCV so we can print it out later
        for (i = 0; i < numVariantsForQueryName; i++)   {
                queryUCV = (LPCTSTR)queryUCVs[i];
                for (j = 0; j < numVariantsForWatchName; j++)   {
                        watchUCV = (LPCTSTR)(*watchUCVs)[j];
                        if (!tds_unsigned_strcmp((const unsigned char *)queryUCV,

=>                      (const unsigned char *)watchUCV))       {
                                shouldMatchOnUCV = TRUE;
                                strcpy(ucvMatchForWatchName, watchUCV);
                                break;
                        }
                }
        }
        delete watchUCVs;
}

*/
```

```
/*
void    CTDSDlg::outputWatchResults()
{

        logstream << endl << "Watch name results:" << endl;
        logstream << "Watch Name: " << watchName << endl;
        logstream << "Should have matched on UCV:  ";

        if (shouldMatchOnUCV)
                logstream << "Yes ---> " << ucvMatchForWatchName << endl;
        else
                logstream << "No" << endl;
        logstream << "Did matched on UCV:  ";
        if (didMatchOnUCV)
                logstream << "Yes" << endl;
        else
                logstream << "No" << endl;

        logstream << "Should have matched on UCV Bitmap:  ";
        if (shouldMatchOnBitmap)
                logstream << "Yes" << endl;
        else
                logstream << "No" << endl;
        logstream << "Did matched on UCV Bitmap:  ";
        if (didMatchOnBitmap)
                logstream << "Yes" << endl;
        else
                logstream << "No" << endl;

        logstream << "Should have passed Edit Distance:  ";
        if (shouldPassEditDist)
                logstream << "Yes. ";
        else
                logstream << "No. ";
        logstream << "Edit Dist Score was:  " << watchNameEditDistanceScore << endl;
        logstream << "Did pass Edit Distance:  ";
        if (didPassEditDist)
                logstream << "Yes" << endl;
        else
                logstream << "No" << endl;

        char                                                    spacedWatchName[TDS_MAX_NAME + 3];
        sprintf(spacedWatchName, " %s ", watchName);
        strupr(spacedWatchName);

        if (shouldMatchOnUCV == false)  {
                int                                                             i;
                vector<CString>         *queryUCVs = NULL;
                int                                                             numVariantsForQueryName;
                tds_vargen_code         genVarRC;

                logstream << endl << "UCV Details:" << endl;

                queryUCVs = ruleSet.genVariants(spacedQueryName,

=>                                      TRUE,           // devowel

=>                                      TRUE,           // translate

=>                                      UCV_LEN_IN_BITMAP,

=>                                      genVarRC);
                numVariantsForQueryName = queryUCVs->size();

                logstream << "Query Generated " << numVariantsForQueryName
                                        << " UCV(S) variants" << endl;
```

```
	for (i = 0; i < numVariantsForQueryName; i++)	{
		logstream << "Variant " << i + 1 << " "
								<< (LPCTSTR) (*queryUCVs) [i] << endl;

	}
	delete queryUCVs;

	vector<CString>		*watchUCVs = NULL;
	int								numVariantsForWatchName;

	watchUCVs = ruleSet.genVariants(spacedWatchName,

						TRUE,		// devowel
=>
						TRUE,		// translate
=>
						UCV_LEN_IN_BITMAP,
=>
						genVarRC);
=>
	numVariantsForWatchName = watchUCVs->size();

	logstream << "Watch name Generated " << numVariantsForWatchName
						<< " UCV(5) variants" << endl;

	for (i = 0; i < numVariantsForWatchName; i++)	{
		logstream << "Variant " << i + 1 << " "
								<< (LPCTSTR) (*watchUCVs) [i] << endl;

	}
	delete watchUCVs;

}

char	watchRegEx[1000 + 1];
char	queryRegEx[1000 + 1];
CNfa	watchNfa;

logstream << endl << "Exact match check:  " << endl;
strcpy(queryRegEx, (LPCTSTR) ruleSet.TranslateWord(spacedQueryName));
logstream << "Query regex is:  " << queryRegEx << endl;

strcpy(watchRegEx, (LPCTSTR) ruleSet.TranslateWord(spacedWatchName));
watchNfa.Set(watchRegEx);
logstream << "Watch regex is:  " << watchRegEx << endl;

if (watchNfa.Match(queryNfa))
	logstream << "Regexes: Match" << endl;
else
	logstream << "No exact Match on Regexes:" << endl;

}
*/

//	??? both these methods should check the rc from genVariants, and have some way
//	to revert to the old edit distance if genVariants could not generate all the variants.
//	Also, we need to write an edit distance that uses the feature table, but does not use
//	SECs.
//	Also, we need to set a flag if the query's variants were not able to be generated in
//	full.  If this is the case, we should not do the edit distance adjustment.
float	TDSSearcher::calcSingVovelBruteForceEditDist(char *dbName, int queryCulture,
							e_tds_culture culture,
=>
							float origScore)
=>
{							spacedDBName[TDS_MAX_NAME + 3];
	char
	tds_vargen_code		genVarRC;
```

```
	float							bestScore = 0.0;

	sprintf(spacedDBName, " %s ", dbName);
	strupr(spacedDBName);

	//	now get the variants for the second name
	vector<CString> *dbNameVariants = ruleSets[culture]->genVariants(spacedDBName, false,

=>
=>		TRUE, 0, genVarRC);

	if (genVarRC == TDS_VARGEN_CODE_OK)	{
		vector<CString> *qNameVariants = querySingVowelVariants[queryCulture];
		int	numDBNameVariants = dbNameVariants->size();
		int	numQNameVariants = qNameVariants->size();
		float	tempScore = 0.0;

		const char *queryVariant;
		const char *dbVariant;

		//	do a cartisian product comparison of the two vectors.
		for (int i = 0; (i < numQNameVariants) && (*stopVar == false); i++)	{
			queryVariant = (LPCTSTR) (*qNameVariants) [i];
			for (int j = 0; (j < numDBNameVariants) && (*stopVar == false); j++)	{
				dbVariant = (LPCTSTR) (*dbNameVariants) [j];

				tempScore = featureEditDistancizer->getDistanceScore((unsigned char *)queryVar
=> iant,

=>						(unsigned char *)dbVariant);
=>				if (tempScore > bestScore)	{
					bestScore = tempScore;
				}
			}
		}
	}
	else	{
		bestScore = origScore;
		if (genVarRC == TDS_VARGEN_CODE_TOO_MANY)	{
			if (logstream)	{
				if (logDebugInfo)
					*logstream << "DB Name " << dbName <<				" generated too many v
=> ariants to do edit distance adjustment" << endl;
			}
			else	{
				*logstream << "DB Name " << dbName <<				" generated an invalid express
=> ion during edit distance adjustment" << endl;
			}
		}
	}

	delete dbNameVariants;

	return bestScore;
}

//	same as calcSingVovelBruteForceEditDist, but for 3 vowel rules
float	TDSSearcher::calc3VovelBruteForceEditDist(char *dbName, int queryCulture,
							e_tds_culture culture,
=>
```

```
=>                                                                  float origScore)
{
        char                                                spacedDBName[TDS_MAX_NAME + 3];
        tds_vargen_code             genVarRC;
        float                                               bestScore = 0.0;

        sprintf(spacedDBName, " %s ", dbName);
        strupr(spacedDBName);

        //      now get the variants for the second name
        vector<CString> *dbNameVariants = ruleSets3V[culture]->genVariants(spacedDBName, false,

=>
=>              TRUE; 0, genVarRC);

        if (genVarRC == TDS_VARGEN_CODE_OK)        {
                vector<CString> *qNameVariants = query3VowelVariants(queryCulture);
                int     numDBNameVariants = dbNameVariants->size();
                int     numQNameVariants = qNameVariants->size();
                float   tempScore = 0.0;

                const char *queryVariant;
                const char *dbVariant;

                //      do a cartisian product comparison of the two vectors.
                for (int i = 0; (i < numQNameVariants) && (*stopVar == false); i++)      {
                        queryVariant = (LPCTSTR)(*qNameVariants)[i];
                        for (int j = 0; (j < numDBNameVariants) && (*stopVar == false); j++)     {
                                dbVariant = (LPCTSTR)(*dbNameVariants)[j];

                                tempScore = featureEditDistancizer->getDistanceScore((unsigned char *)queryVar
=> iant,

=>
=>                                      (unsigned char *)dbVariant);
                                if (tempScore > bestScore)       {
                                        bestScore = tempScore;
                                }
                        }
                }
        }
        else    {
                bestScore = origScore;
                if (genVarRC == TDS_VARGEN_CODE_TOO_MANY)        {
                        if (logstream)  {
                                if (logDebugInfo)
                                        *logstream << "DB Name " << dbName <<
                                                                                  " generated too many v
=> ariants to do edit distance adjustment" << endl;
                        }
                        else    {
                                *logstream << "DB Name " << dbName <<
                                                                          " generated an invalid express
=> ion during edit distance adjustment" << endl;
                        }
                }
        }

        delete dbNameVariants;

        return bestScore;
}


//      function to see if the supplied database name is an
//      exact phonetic match with the query name.  We use
```

```
//      the member CNfa object to complete the check
//
//      narrative paragraph number 4.2.3
bool    TDSSearcher::checkExactPhonMatch(char *dbName, e_tds_culture culture, int queryCultureIndex)
{
        char    dbRegEx[1000 + 1];
        CNfa    dbNfa;
        char    spacedDbName[TDS_MAX_NAME + 3];


        sprintf(spacedDbName, " %s ", dbName);
        strupr(spacedDbName);
        strcpy(dbRegEx, (LPCTSTR)ruleSets3V[culture]->TranslateWord(spacedDbName));
        dbNfa.Set(dbRegEx);

        return dbNfa.Match(queryNfa[queryCultureIndex]);
}


void TDSSearcher::convertVar2Grp(LPCSTR variant, CString *group, bool skipRepeatChars)
{
        unsigned char prevVariantChar = 0;


        char    *groupStringPtr = group->GetBuffer(TDS_MAX_NAME + 1);
  while (*variant)      {
                if ((skipRepeatChars == false) || ((unsigned char)*variant != prevVariantChar)) {
                        *groupStringPtr = m_group_array[(unsigned char)*variant];
                        prevVariantChar = (unsigned char)*variant;
                        groupStringPtr++;
                }
                variant++;
        }
        *groupStringPtr = '\0';
        group->ReleaseBuffer();
}


//      queryCultureIndex is a value of 0 or 1, depending on which culture we are working
//      on (remember that our query variables must handle up to 2 cultures).
void    TDSSearcher::calcQueryIPAInfo(int queryCultureIndex, e_tds_culture culture)
{
        int                                                          i, k;
        unsigned char                   *varString;
        vector<CString>         *qVariants = querySingVowelVariants(queryCultureIndex);
        int                                                          numVariants = qVariants->size();
        char                                                 catString[2];


        //      clear out the start IPA characters
        queryStartIPAChars[queryCultureIndex][0] = '\0';
        queryStartVowelFlag[queryCultureIndex] = ' ';

        catString[1] = EOS;
        for ( i = 0; i < numVariants; i++ )      {
                k = 0;
                varString = (unsigned char *)(LPCTSTR)(*qVariants)[i];
                if (*varString != EOS)  {

                        if (IS_VOWEL(varString[0]))      {
                                //      starting with vowel
                                if (queryStartVowelFlag[queryCultureIndex] == ' ')
                                        queryStartVowelFlag[queryCultureIndex] = 'Y';
                                else
                                        if (queryStartVowelFlag[queryCultureIndex] == 'N')
                                                queryStartVowelFlag[queryCultureIndex] = 'B';
```

```
				}
				else	{		//	starting with consonant
					if (queryStartVowelFlag[queryCultureIndex] == ' ')
						queryStartVowelFlag[queryCultureIndex] = 'N';
					else
						if (queryStartVowelFlag[queryCultureIndex] == 'Y')
							queryStartVowelFlag[queryCultureIndex] = 'B';
				}

				while ((*varString != EOS) && (IS_VOWEL(*varString)))
					varString++;
				if (*varString != EOS)	{
					if (strchr((const char *)queryStartIPAChars[queryCultureIndex], *varString) ==
 => NULL)	{
						catString[0] = *varString;
						strcat((char *)queryStartIPAChars[queryCultureIndex], (const char *)ca
 => tString);
					}
				}
			}
		}
}


//	generate the single vowel variants for the query, using the specified
//	culture's rules.
bool	TDSSearcher::genQueryVariantsForSingVowel(int queryCultureIndex, e_tds_culture culture,
 =>					char *queryRegEx)
{
	bool						rc = true;
	int								numVariantsForQueryName;
	tds_vargen_code		genVarRC;
	vector<CString>		*tempVariants = ruleSets[culture]->genVariants(queryRegEx,
 =>											false,
 =>		/* devowel */
 =>											false,
 =>		/* translate */
 =>											0,
 =>				/* truncate */
 =>											genVarR
 => C);

	//	reset truncation flag
	singVowelVariantsTrunced[queryCultureIndex] = false;

	numVariantsForQueryName = tempVariants->size();
	querySingVowelVariants[queryCultureIndex] = tempVariants;
	if (genVarRC != TDS_VARGEN_CODE_BAD_REGEX)	{
		if (logstream != NULL)
			*logstream << endl << "Culture " << cultureStrings[culture]
							<< ": Query Generated " << numVariantsForQuery
 => Name
							<< " single vowel variants" << endl;
		//	spit out the variants if debugging is turned on
		if (logDebugInfo)	{
			for (int j = 0; j < numVariantsForQueryName; j++)	{
				*logstream << "Variant " << j + 1 << ": "
								<< (LPCTSTR)(*tempVariants)[j] << endl
 => ;
			}
		}
```

```
	}
	if (genVarRC != TDS_VARGEN_CODE_OK)	{
		if (genVarRC == TDS_VARGEN_CODE_BAD_REGEX)	{
			sprintf(errMsg, "Query generated an invalid regular expression (single vowel): %s"
										"	Check the
 =>  error log file for details", queryRegEx);
			if (logstream)
				*logstream << errMsg << endl;
			rc = false;
		}
		else	{
			if (genVarRC == TDS_VARGEN_CODE_TOO_MANY)	{
				//	note that we could not generate all the variants, so that we will skip
				//	brute force adjustment even if we were requested to do it.
				singVowelVariantsTrunced[queryCultureIndex] = true;

				//	not an error, but put a warning in the log file
				if (logstream)
					*logstream << "Warning: Query generated too many single vowel variants
 => , "
										"Query IPA inf
 => o may be incomplete." << endl;
			}
		}
	}

	return rc;
}


//	generate the single vowel variants for the query, using the specified
//	culture's rules.
bool	TDSSearcher::genQueryVariantsFor3Vowel(int queryCultureIndex, e_tds_culture culture,
 =>					char *queryRegEx)
{
	bool						rc = true;
	int								numVariantsForQueryName;
	tds_vargen_code		genVarRC;
	vector<CString>		*tempVariants = ruleSets3V[culture]->genVariants(queryRegEx,
 =>											false,
 =>		/* devowel */
 =>											false,
 =>		/* translate */
 =>											0,
 =>				/* truncate */
 =>											genVarR
 => C);

	//	reset truncation flag
	threeVowelVariantsTrunced[queryCultureIndex] = false;
	numVariantsForQueryName = tempVariants->size();
	query3VowelVariants[queryCultureIndex] = tempVariants;
	if (genVarRC != TDS_VARGEN_CODE_BAD_REGEX)	{
		if (logstream != NULL)
			*logstream << endl << "Culture " << cultureStrings[culture]
							<< ": Query Generated " << numVariantsForQuery
 => Name
							<< " three vowel variants" << endl;
	}
	if (genVarRC == TDS_VARGEN_CODE_OK)	{
		//	spit out the variants if debugging is turned on
		if (logDebugInfo)	{
			if (logstream != NULL)	{
				for (int j = 0; j < numVariantsForQueryName; j++)	{
					*logstream << "Variant " << j + 1 << ": "
										<< (LPCTSTR)(*tempVariants)[j]
»	<< endl;
				}
			}
		}
	}
	else	{
		if (genVarRC == TDS_VARGEN_CODE_BAD_REGEX)	{
			sprintf(errMsg, "Query generated an invalid regular expression under 3V rules: %s"
										"	Check the
»	error log file for details", queryRegEx);
			if (logstream)
				*logstream << errMsg << endl;
			rc = false;
		}
		else	{
			if (genVarRC == TDS_VARGEN_CODE_TOO_MANY)	{
				//	note that we could not generate all the variants, so that we will skip
				//	brute force adjustment even if we were requested to do it.
				threeVowelVariantsTrunced[queryCultureIndex] = true;
				//	not an error, but put a warning in the log file
				if (logstream)
					*logstream << "Warning: Query generated too many three vowel variants,
»	"
										"Query IPA inf
»	o may be incomplete" << endl;
			}
		}
	}

	return rc;
}


void	TDSSearcher::computeQueryGroup(int queryCultureIndex, e_tds_culture culture)
{
	vector<CString>		*qVariants = querySingVowelVariants[queryCultureIndex];
	int						numVariants = qVariants->size();
	int						j;


	//	now generate groups
	//	start the CString that will hold the groups out at the
	//	size of the biggest name.  The variants can grow bigger than
	//	this, but its a good place to start.
	CString tempGroupString((TCHAR)0, TDS_MAX_NAME + 1);
	for (j = 0; j < numVariants; j++)	{
		convertVar2Grp((LPCTSTR)(*qVariants)[j], &tempGroupString,
										true /*skip repeat cha
»	rs */);
		queryGroups[queryCultureIndex].insert(tempGroupString);
	}
	//	Store in queryStats the number of distinct groups for this culture
	queryStats.numQueryGroups[queryCultureIndex] = queryGroups[queryCultureIndex].size();

	//	write out the number of distinct groups, which may be smaller
	//	than the number of variants (two variants can produce the
	//	same group.
	if (logstream != NULL)	{
		*logstream << endl << "Culture " << cultureStrings[culture]
							<< ": Query Generated " << queryStats.numQueryGroups[q
»	ueryCultureIndex]
```

```
							<< " groups" << endl;
		if (logDebugInfo)	{
			set<CString>::iterator	groupStringIter;

			j = 0;
			for (groupStringIter = queryGroups[queryCultureIndex].begin();
						groupStringIter != queryGroups[queryCultureIndex].end();
						groupStringIter++)	{
				j++;
				*logstream << "Group " << j << ": "
										<< (LPCTSTR)(*groupStringIter) << endl
»	;
			}
		}

	}

}


//	??? this function should be moved to the library at some point.
// reads in the file that initializes the m_group_array which contains
// the information which is used to translate from variant to group
// these groups are determined by the linguists, Z denotes an unused
// index, Z should not occur in any group.
bool TDSSearcher::initGroupArray(char *groupArrayFileName, char *aGroupArray)
{
  FILE  *ga;
  char  linebuff[255 + 1];
  int		index;
  char  value;
	bool	rc = true;
	int		lineNo = 1;

	//	init the entire array with 'Z'.
	for (int i = 0; i < 256; i++)
		aGroupArray[i] = 'Z';
  if ((ga = fopen(TDS_GROUP_ARRAY_FILE, "r")) == NULL)  {
		sprintf(errMsg, "Could not open the Group Array file %s", groupArrayFileName);
		if (logstream != NULL)
			*logstream << errMsg << endl;
   rc = false;
  }
	else	{
		while (fgets(linebuff, 255, ga) != NULL)	{ // get the next line
			//	get the first two items on the line (the index and the value)
			sscanf(linebuff, "%d %c", &index, &value);
			//	make sure index is valid
			if ((index > 255) || (index < 0))	{
				sprintf(errMsg, "Group Array file %s, Line %d, invalid.index %d",
										groupArrayFileName,
										lineNo,
										index);
			if (logstream != NULL)
				*logstream << errMsg << endl;
				rc = false;
				break;
			}
			aGroupArray[index] = value; // assign the value
			lineNo++;
		}
		//	make sure that the reason fgets failed is that we are at the end of
		//	the file.  Otherwise, there was a problem.
		if (feof(ga) == 0)	{
			sprintf(errMsg, "Error reading from Group Array file %s, Line %d",
									groupArrayFileName,
                                                                lineNo);
                        if (logstream != NULL)
                                *logstream << errMsg << endl;
                        rc = false;
                }
          fclose(ga);
        }

  return rc;
}


bool    TDSSearcher::setMaxNamesToReturnPerQuery(int anInt)
{
        bool    rc;


        if (anInt > 0)  {
                maxNamesToReturnPerQuery = anInt;
                rc = true;
        }
        else    {
                rc = false;
                sprintf(errMsg, "Invalid MaxNamesToReturnPerQuery value %d", anInt);
                if (logstream != NULL)
                        *logstream << errMsg << endl;
        }
        return rc;
}


void    TDSSearcher::setWatchName(const char *aWatchName)
{
        if (aWatchName == NULL)
                watchName[0] = '\0';
        else    {
                strncpy(watchName, aWatchName, TDS_MAX_NAME);
                watchName[TDS_MAX_NAME] = '\0';
        }
}


void    TDSSearcher::setRankerParms(RParameters *aRankerParms)
{
        if (aRankerParms == NULL)
                rankerParms = &defaultRankerParms;
        else
                rankerParms = aRankerParms;
}


void    TDSSearcher::setCultureInfo(e_tds_cultureMode aCultMode,

  **                            e_tds_culture aSpecifiedCult)
{
        cultureMode = aCultMode;
        specifiedCulture = aSpecifiedCult;
}


//      function to get the exact matches, which the ranker is holding.  We
//      copy the pointers to the ExtRName object from the ranker to
//      a vector that is passed to us.  Note that since the exact matches
//      always come first, and need no edit distance adjustment, that there
//      is no need to check and process the fat ranker (if we were using one).
//      Exact matches go directly to the final ranker, and never to the
```

```
//      fat ranker.
void    TDSSearcher::getResultNamesForExactMatch(vector<ExtRName *> &resultVector)
{
tds_results_iterator_t          rankerIterator;
ExtRName                                                                *extResultName;

        for (rankerIterator = ranker.m_scoredNames.rbegin();
                                rankerIterator != ranker.m_scoredNames.rend();
                                rankerIterator++)       {
                extResultName = (ExtRName       *)(*rankerIterator);
                //      we only want exact matches
                if (extResultName->getIsExact() == false)
                        break;
                else    {
                        resultVector.push_back(extResultName);
                }
        }
}


void    TDSSearcher::getResultNamesForSimilarMatch(vector<ExtRName *> &resultVector)
{
tds_results_iterator_t          rankerIterator;
ExtRName                                                                *extResultName;


        //      make sure that if we are using a fat ranker, those names get processed
        //      this gets done automatically from the similar match, but if they only
        //      asked for exact, we need to do it here.
        if (postRankerBFMode != TDS_BF_MODE_NONE)       {
                if (fatRankerProcessed == false)
                        processFatRankerResults();
        }

//      first go past the exact matches
        for (rankerIterator = ranker.m_scoredNames.rbegin();
                                rankerIterator != ranker.m_scoredNames.rend();
                                rankerIterator++)       {
                extResultName = (ExtRName       *)(*rankerIterator);
                //      we only want exact matches
                if (extResultName->getIsExact() == false)
                        break;
        }

        //      now that we are past the exacts, copy the pointers
        for (;
                                rankerIterator != ranker.m_scoredNames.rend();
                                rankerIterator++)       {
                extResultName = (ExtRName       *)(*rankerIterator);
                resultVector.push_back(extResultName);
                //      resultVector.push_back((ExtRName        *)(*rankerIterator));
        }
}


void    TDSSearcher::getAllResultsNames(vector<ExtRName *> &resultVector)
{
tds_results_iterator_t          rankerIterator;


        for (rankerIterator = ranker.m_scoredNames.rbegin();
                                rankerIterator != ranker.m_scoredNames.rend();
                                rankerIterator++)       {
                resultVector.push_back((ExtRName        *)(*rankerIterator));
        }
}
```

```
//      delete the NameRecord values that are in the names map.
//      since we are using pointers to NameRecord objects
void    TDSSearcher::emptyNamesMap()
{
        name_record_map_t::iterator     mapIter;

        mapIter = namesMap.begin();
        for (mapIter = namesMap.begin(); mapIter != namesMap.end(); mapIter++)  {
                delete mapIter->second;
        }
        namesMap.clear();
}

//      returns the number of results we are currently holding
//      If we are using a fat ranker, we should make sure that we have
//      already retreived the names from the Fat Ranker, adjusted their
//      scores, and sent them to the final Ranker
int             TDSSearcher::getNumResultNames()
{
        return ranker.nameCount();
}


//      get the names from the fat ranker, adjust their edit distance,
//      and submit them to the final ranker.  Note that we need to make
//      copies of the ExtRName objects, since the ranker owns those.
//      If we do not make a copy, both rankers will have a pointer to the
//      same object, and both will try to delete it.
//
//      narrative paragraph number 4.5.1
//      narrative paragraph number 4.5.2
//      narrative paragraph number 4.5.3
void    TDSSearcher::processFatRankerResults()
{
tds_results_iterator_t          rankerIterator;
ExtRName                                                                        *tempFatName;
ExtRName                                                                        *tempFinalName;
float                                                                                   editDistScore;
int                                                                                             queryCultureIn
  => dex;


        for (rankerIterator = fatRanker.m_scoredNames.rbegin();
                                (rankerIterator != fatRanker.m_scoredNames.rend()) && (*stopVar == false);
                                rankerIterator++)       {
                tempFatName = (ExtRName *)(*rankerIterator);

                //      if the culture of the name from the fat ranker is ANGLO, we will
                //      be doing brute force acording to the anglo rules, which always
                //      appear at index 0.  The other culture (if one was used) is always
                //      at index 1.
                if (tempFatName->getPipeCulture() == TDS_CULT_ANGLO)
                        queryCultureIndex = 0;
                else
                        queryCultureIndex = 1;

                if (postRankerBFMode == TDS_BF_MODE_SINGLE)
                        editDistScore = calcSingVowelBruteForceEditDist((char *)tempFatName->getStr().c_str(),

  =>
  =>  queryCultureIndex,

  =>
  =>  tempFatName->getPipeCulture(),
```

```
  =>
  =>  tempFatName->getPhoneticScore());
                else
                        editDistScore = calc3VowelBruteForceEditDist((char *)tempFatName->getStr().c_str(),

  =>
  =>  queryCultureIndex,

  =>
  =>  tempFatName->getPipeCulture(),

  =>
  =>  tempFatName->getPhoneticScore());

                tempFinalName = new ExtRName(*tempFatName);
                tempFinalName->setPhoneticScore(editDistScore);

                ranker.submit(tempFinalName);
        }
        fatRankerProcessed = true;
}
```

```
// UCVBitMap.cpp : implementation file
//
//      Copyright (C) 1998, Language Analysis Systems Inc.
//

#include        <sys/types.h>
#include        <sys/stat.h>

#include        <iostream>
#include <stdio.h>

using namespace std;


#include "stdafx.h"
#include "UCVBitMap.hpp"
#include "tds_util.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


UCVBitMap::UCVBitMap(const char *fileName, BOOL useMemory)
{
        //      stat the file and get the file size
        //      from that, use sizeof(bitmap_entry_t) to
        //      determine how many map_entry_t objects
        //      we should allocate.  (Note that the file size
        //      should be evenly divisible by the size of the structure,
        //      where the structure includes the variable length ucvKey,
        //      (not just the char placeholder).

        //      initialize stuff.
        mapData = NULL;
        ucvBitMapFile = NULL;
        numMapEntries = 0;
        status = FALSE;
        usingMemory = useMemory;                //      save the mode the requested

        //      stat the file for the file size
  struct stat statBuf;
        if (stat(fileName, &statBuf ) == 0)
                numMapEntries = statBuf.st_size / sizeof(bitmap_entry_t);
        else
                AfxMessageBox("Could not stat the BitMap file");

        //      if we found a file with some map entries
        if (numMapEntries > 0)
   {
     ucvBitMapFile = fopen(fileName, "rb");

     if (ucvBitMapFile == NULL)
     {
       CString msg;
       msg.Format("Could not open UCV Map file %s", fileName);
       AfxMessageBox(msg);
     }
                else
     {
                        if (useMemory)
       {
                                mapData = (bitmap_entry_ptr_t) new bitmap_entry_t[numMapEntries];
                                if (mapData != NULL)
          {
                                        int     bytesRead = fread((char *)mapData, 1, statBuf.st_size, ucvBitM
  => apFile);
                                        if (bytesRead != statBuf.st_size)
            {
                                                CString msg;
                                                msg.Format("Error: only read %d bytes from UCV BitMap file %s"
  => , bytesRead, fileName);
                                                AfxMessageBox(msg);
                                        }
                                        else
                                                status = TRUE;
                                }                       //      map data not NULL
                        }               //      using memory
                else
                        status = TRUE;
                //      close the file if there was a problem, or if we are loading into memory
                if ((status == FALSE) || (useMemory == TRUE))
     {
                        fclose(ucvBitMapFile);
                        ucvBitMapFile = NULL;
                }
          }             //      good file open
        }
        if (status == TRUE)
                buildBitTable();
}

UCVBitMap::~UCVBitMap()
{
        if (mapData != NULL)
                delete [] mapData;
        if (ucvBitMapFile != NULL)
                fclose(ucvBitMapFile);
}


//      function to search through the entire map of bitmap values,
//      and test each 32 bit value in the map against each of the
//      32 bit values found in the variantBitVals vector.
//      For each map entry that matches according to some threshold,
//      we get the indexOffset and numIndexEntries values, and use
//      them to collect name offsets (into the names file).  We
//      add these offsets (pending uniqueness) to the vector that
//      we return at the end of the function.
//      NOTE: this function allocates a vector that must be freed
//      by the caller.
set<unsigned int> *     UCVBitMap::search(vector<bits_64_t> *variantBitVals,

  =>                                    FILE *ucvIndexFile, BOOL *cancelFlag)
{
        int                                                                     numVariantBitV
  => als = variantBitVals->size();
        int                                                                     mapIndex;
        int                                                                     queryVariantIn
  => dex;
        int                                                                     indexFileIndex
  => ;
        int                                                                     numIndexValues
  => ForThisUCV;
        unsigned int                                    offsetInNamesFile;
        set<unsigned int>                       *nameOffsets;
        //int                                                                   k;
        int                                                                     numNameOffsets
  => ForThisQueryName = 0;
        //BOOL                                                          isDup;
```

```
    vector<unsigned int>          queryVariantNumBits;
    unsigned char                                        *bytePtr;
    bits_64_t                                                            tempVarantBitVal;
    int                                                                                    mapVariantNumB
=> its;

    //      create the vector that we will return.  This holds offsets into the names file.
    nameOffsets = new set<unsigned int>;

    //      up front, calcululate the number of bits that are turned on in each of the
    //      query variants.  We need this value when doing a bitwise comparison, and it
    //      is a relatively expensive computation to be doing for each time we compare it
    //      to a map value.
    for (queryVariantIndex = 0; queryVariantIndex < numVariantBitVals; queryVariantIndex++) {
            tempVarantBitVal = (*variantBitVals)[queryVariantIndex];
            bytePtr = (unsigned char *)&tempVarantBitVal;
            //      queryVariantNumBits.push_back(bitTable[bytePtr[0]] + bitTable[bytePtr[1]] +
            //                                                                     bitTable[byteP
=> tr[2]] + bitTable[bytePtr[3]]);
            queryVariantNumBits.push_back(bitTable[bytePtr[0]] + bitTable[bytePtr[1]] +
                                                                     bitTable[bytePtr[2]] + bitTabl
=> e[bytePtr[3]] +
                                                                     bitTable[bytePtr[4]] + bitTabl
=> e[bytePtr[5]] +
                                                                     bitTable[bytePtr[6]] + bitTabl
=> e[bytePtr[7]]);
    }

    for (mapIndex = 0; (mapIndex < numMapEntries) && (*cancelFlag == FALSE); mapIndex++)     {
            bytePtr = (unsigned char *)&mapData[mapIndex].bitMapVal1;
            //      mapVariantNumBits = bitTable[bytePtr[0]] + bitTable[bytePtr[1]] +
            //                                                                     bitTable[bytePtr[2]] +bitTable
=> [bytePtr[3]];
            mapVariantNumBits = bitTable[bytePtr[0]] + bitTable[bytePtr[1]] +
                                                                     bitTable[bytePtr[2]] + bitTabl
=> e[bytePtr[3]] +
                                                                     bitTable[bytePtr[4]] + bitTabl
=> e[bytePtr[5]] +
                                                                     bitTable[bytePtr[6]] + bitTabl
=> e[bytePtr[7]],
            for (queryVariantIndex = 0; queryVariantIndex < numVariantBitVals; queryVariantIndex++) {
                    if (bitCompare((*variantBitVals)[queryVariantIndex], &mapData[mapIndex].bitMapVal1,
                                                                               queryVariantNumBits[qu
=> eryVariantIndex] + mapVariantNumBits))     {
                            //      in the index file, seek to the position specified by
                            //      the mapData[mapIndex].indexOffset position, and for
                            //      mapData[mapIndex].numIndexEntries, read an offset

                            if (fseek(ucvIndexFile, mapData[mapIndex].indexOffset * sizeof(unsigned int),
=> SEEK_SET) == 0)    {
                                    numIndexValuesForThisUCV = mapData[mapIndex].numIndexEntries;
                                    for (indexFileIndex = 0; indexFileIndex < numIndexValuesForThisUCV; in
=> dexFileIndex++)    {
                                            //      should check for good read below
                                            fread((char *)&offsetInNamesFile, sizeof(unsigned int), 1, ucv
=> IndexFile);

                                            //      add (in a unique fashion) the offset found  in the ind
=> ex file
                                            nameOffsets->insert(offsetInNamesFile);

                                            /*
                                            isDup = FALSE;
                                            for (k = 0; k < numNameOffsetsForThisQueryName; k++)      {
                                                    if ((*nameOffsets)[k] == offsetInNamesFile)      {
                                                            isDup = TRUE;
```

```
                                                            break;
                                                    }
                                            }
                                            if (isDup == FALSE)      {
                                                    nameOffsets->push_back(offsetInNamesFile);
                                                    numNameOffsetsForThisQueryName++;
                                            }
                                            */
                                    }
                            }
                            else     {
                                    char    errMsg[1000];

                                    sprintf(errMsg, "Error seeking to offset %d in index file", mapData[ma
=> pIndex].indexOffset * sizeof(unsigned int));
                                    AfxMessageBox(errMsg);
                                    break;
                            }
                    }
            }
    }

    return  nameOffsets;
}


//      build a table that says how many bits a byte value
//      has turned off.
void    UCVBitMap::buildBitTable()
{
        unsigned char   tempByte;
        int                                     i, j;
        unsigned char   numberOfBits;

        for (i = 0; i < 256; i++)       {
                tempByte = i;
                numberOfBits = 0;
                for (j = 0; j < 8; j++) {
                        if (tempByte & 1)                               //      use this when array says how m
=> any 1's
                        //      if ((tempByte & 1) == 0)                //      use this when array says how m
=> any 0's
                                numberOfBits++;
                        tempByte >>= 1;
                }
                bitTable[i] = numberOfBits;
        }
}

//      BOOL    UCVBitMap::bitCompare(unsigned int bitVal1, unsigned int bitVal2, int bitsThatCouldHaveMatched
=> )
BOOL    UCVBitMap::bitCompare(bits_64_t bitVal1, unsigned int *bitVal2, int bitsThatCouldHaveMatched)
{
        BOOL    rc = TRUE;
        unsigned        int             maskedValue[2];
        unsigned char           *bytePtr;

        maskedValue[0] = bitVal1.int1 & bitVal2[0];
        maskedValue[1] = bitVal1.int2 & bitVal2[1];
        bytePtr = (unsigned char *)&maskedValue;

        //      the maskedValue contains 0 for each bit position that matched, and
        //      1 for each that did not.  We break this value up into 4 sections,
        //      one byte each, and consult our bitTable to get the number of 0 bits
        //      the value contains
        /*
```

```
	unsigned char	byte1 = bytePtr[0];
	unsigned char	byte2 = bytePtr[1];
	unsigned char	byte3 = bytePtr[2];
	unsigned char	byte4 = bytePtr[3];

	byte1 = bitTable[byte1];
	byte2 = bitTable[byte2];
	byte3 = bitTable[byte3];
	byte4 = bitTable[byte4];

	int totalBits = byte1 + byte2 + byte3 + byte4;
	*/

	int totalBits =  bitTable[bytePtr[0]] + bitTable[bytePtr[1]] +
												bitTable[bytePtr[2]] +bitTable
=> [bytePtr[3]] +
												bitTable[bytePtr[4]] + bitTabl
=> e[bytePtr[5]] +
												bitTable[bytePtr[6]] +bitTable
=> [bytePtr[7]];
	if (bitsThatCouldHaveMatched == 0)
		return FALSE;
	else	{
		totalBits *= 2;		//	give credit for both bits that matched
		return ((float)totalBits / (float)bitsThatCouldHaveMatched) >= threshold;
	}
}
```

```
// UCVMap.cpp : implementation file
//
//      Copyright (C) 1998, Language Analysis Systems Inc.

#include        <sys/types.h>
#include        <sys/stat.h>

#include <use_ansi.h>
#include <fstream>
#include        <iostream>
#include <stdio.h>

using namespace std;


#include "stdafx.h"
#include "UCVMap.hpp"
#include "tds_util.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


UCVMap::UCVMap(const char   .   *fileName, BOOL useMemory, int ucv_length)
{
        //      stat the file and get the file size
        //      from that, use sizeof(map_entry_t) to
        //      determine how many map_entry_t objects
        //      we should allocate.  (Note that the file size
        //      should be evenly divisible by the size of the structure,
        //      where the structure includes the variable length ucvKey,
        //      (not just the char placeholder).

        //      initialize stuff.
        mapData = NULL;
        ucvMapFile = NULL;
        numMapEntries = 0;
        status = FAISE;
        usingMemory = useMemory;                //      save the mode the requested
        ucvLen = ucv_length;
        mapItemSize = sizeof(map_entry_t) + ucv_length;

        //      offset of key within the structure is the size of the structure
        //      minus 1.  This is true so long as the char placeholder is the last
        //      item in the structure.
        ucvKeyOffsetInStructure = sizeof(map_entry_t) - 1;

        //      stat the file for the file size
  struct stat statBuf;
        if (stat(fileName, &statBuf ) == 0)
  {
                //      the size of an entry is the size of the structure
                //      plus the size of the specified ucv length.  The char
                //      placeholder accounts for the NULL terminator
                if ((statBuf.st_size % mapItemSize) == 0)
                        numMapEntries = statBuf.st_size / mapItemSize;
                else
                        AfxMessageBox("Error: Map file is not divisible by sizeof map entry");
        }
        else
                AfxMessageBox("Could not stat the map file");

        //      if we found a file with some map entries
```

```
        if (numMapEntries > 0)
  {
    ucvMapFile = fopen(fileName, "rb");

    if (ucvMapFile == NULL)
    {
      CString msg;
      msg.Format("Could not open UCV Map file %s", fileName);
      AfxMessageBox(msg);
    }
                else
    {
                        if (useMemory)
      {
                                mapData = (map_entry_ptr_t) new char[statBuf.st_size];
                                if (mapData != NULL)
        {
                                        int     bytesRead = fread((char *)mapData, 1, statBuf.st_size, ucvMapF
=> ile);
                                        if (bytesRead != statBuf.st_size)
          {
                                                CString msg;
                                                msg.Format("Error: only read %d bytes from UCV Map file %s", b
=> ytesRead, fileName);
                                                AfxMessageBox(msg);
                                        }
                                        else
                                                status = TRUE;
                                }                       //      map data not NULL
                        }               //      using memory
                        else
                                status = TRUE;
                        //      close the file if there was a problem, or if we are loading into memory
                        if ((status == FALSE) || (useMemory == TRUE))
      {
                                fclose(ucvMapFile);
                                ucvMapFile = NULL;
                        }
                }               //      good file open
        }
}

UCVMap::~UCVMap()
{
        if (mapData != NULL)
                delete [] mapData;
        if (ucvMapFile != NULL)
                fclose(ucvMapFile);
}

map_entry_t     *               UCVMap::getMapEntryForUCV(const char    *ucv)
{
        map_entry_t             *returnMapEntry = NULL;
        BOOL                            done = FALSE;
        unsigned int    top = 0;
        unsigned int    bottom = numMapEntries - 1;
        unsigned int    testPosition;
        int                                     compRetCode;
        int                                     jumpOfOne = 0;
        char                            *tempString;


        while (!done)   {
                testPosition = top + ((bottom - top) / 2);

                tempString = ((char *)mapData) + (testPosition * mapItemSize) + ucvKeyOffsetInStructure;
                compRetCode = tds_unsigned_strcmp((const unsigned char *)tempString,

                                        (const unsigned char *)ucv);
=>
                if (compRetCode == 0)   {
                        returnMapEntry = (map_entry_ptr_t)(((char *)mapData) + (testPosition * mapItemSize));
                        done = TRUE;
                }
                else    {
                        if (top >= bottom)
                                done = TRUE;
                        else    {
                                if (compRetCode < 0)    {
                                        //      map value is less than the one we are looing for, so look furt
=> her
                                        //      into the map.  That is, make the top the test position
                                        if (top == testPosition)
                                                top += 1;
                                        else
                                                top = testPosition;

                                }
                                else    {
                                        //      map value is greater than the one we are looing for, so look b
=> ackwards
                                        //      into the map.  That is, make the bottom the test position
                                        if (bottom == testPosition)
                                                bottom += 1;
                                        else
                                                bottom = testPosition;
                                }
                        }
                }
        }
        return  returnMapEntry;
}

/*
/////////////////////////////////////////////////////////////////
//
//  This function has been commented out because I moved a
//  copy of it to the filesearch.cpp file.  This was done
//  because the Visual C++ 5.0 compiler was producing erroneous
//  code for the file based search when optimizaiton was turned on
//  since this was only occuring for the file based search we
//  decided to put the code in a separate file so that only that little
//  section would be unoptimized.
//
/////////////////////////////////////////////////////////////////

//      this version does the same thing as above, but it operates
//      from a file rather than from memory
BOOL            UCVMap::getMapEntryForUCV(const char    *ucv, map_entry_t       *returnMapEntry)
{
        BOOL                            done = FALSE;
        BOOL                            found = FALSE;
        unsigned int    top = 0;
        unsigned int    bottom = numMapEntries - 1;
        unsigned int    testPosition;
        int                                     compRetCode;
        int                                     jumpOfOne = 0;
        int                                     bytesRead;
        char                            *tempString;

        while (!done)   {
                testPosition = top + ((bottom - top) / 2);
                if (fseek(ucvMapFile, testPosition * mapItemSize, SEEK_SET) == 0)       {
```

```
                        bytesRead = fread((char *)returnMapEntry, 1, mapItemSize, ucvMapFile);
                        if (bytesRead != mapItemSize)   {
                                CString msg;
                                msg.Format("Error: only read %d bytes from UCV Map file.  Wanted %d",
                                                                                        bytesRead, mapItemSize
=> );

                                AfxMessageBox(msg);
                        }
                        tempString = ((char *)returnMapEntry) + ucvKeyOffsetInStructure;
                        compRetCode = tds_unsigned_strcmp((const unsigned char *)tempString,

=>                                              (const unsigned char *)ucv);
                        if (compRetCode == 0)   {
                                found = TRUE;
                                done = TRUE;
                        }
                        else    {
                                if (top >= bottom)
                                        done = TRUE;
                                else    {
                                        if (compRetCode < 0)    {
                                                //      map value is less than the one we are looing for, so l
=> ook further
                                                //      into the map.  That is, make the top the test position
                                                if (top == testPosition)
                                                        top += 1;
                                                else
                                                        top = testPosition;

                                        }
                                        else    {
                                                //      map value is greater than the one we are looing for, s
=> o look backwards
                                                //      into the map.  That is, make the bottom the test posit
=> ion
                                                if (bottom == testPosition)
                                                        bottom += 1;
                                                else
                                                        bottom = testPosition;
                                        }
                                }
                        }
                }
                else    {
                        char    errMsg[1000];

                        sprintf(errMsg, "Error seeking to offset %d in map file", testPosition * mapItemSize);
                        AfxMessageBox(errMsg);
                        break;
                }
        }

        return  found;
}
*/

//      function to get the index file offset and number of index for the
//      specified UCV.  We do this by searching the map data encapslated
//      by this object.  We support searching the map in either memory
//      of from a file.  For the file case, we must create a map entry
//      structure to hold the reads from disk.  Note that our structure
//      for map_entry_t is somewhat non-standard in that it is actually
//      less than the actual size of a map entry.  We did this to accomodate
//      variable sized UCV key lengths.  For this reason, the stack based
//      entry we create here is actually an array of bytes that is long
//      enough to hold any size UCV we might possibly use (we only go up to 20).
//
BOOL    UCVMap::getMapInfoForUCV(const char *ucv, int *indexOffset, int *numIndexEntries)
{
        BOOL    rc;
        map_entry_t     *mapEntry;
        char                            stackBasedMapEntry[100];

        if (usingMemory)
                mapEntry = getMapEntryForUCV(ucv);
        else    {
                //      from file. so use the stack based map entry to store the
                //      record we are looking for.  If we do not find it, set the
                //      pointer to NULL so it conforms the the check below.
                mapEntry = (map_entry_t *)&stackBasedMapEntry;
                if (getMapEntryForUCV(ucv, mapEntry) == FALSE)
                        mapEntry = NULL;
        }

        if (mapEntry != NULL)   {
                *indexOffset = mapEntry->indexOffset;
                *numIndexEntries = mapEntry->numIndexEntries;
                rc = TRUE;
        }
        else
                rc = FALSE;

        return rc;
}
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   approx.h: interface for the approx class.

   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/////////////////////////////////////////////////////////////////

#ifndef _APPROX_H
#define _APPROX_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef byte
typedef unsigned char byte;
#endif

/////////////////////////////////////////////////////////////////
// these were defined in the original scorer.h file by Robert
// I have put them here so we can use the approx class in other
// programs.  Im sure there is a better way to do this

#ifndef EOS
#define EOS '\0'
#endif

#ifndef NAME_SIZE
#define NAME_SIZE 30 // used to be 16
#endif

#ifndef BLANK
#define BLANK ' '
#endif
/////////////////////////////////////////////////////////////////

class CApprox
{
private:
  bool m_good;
  int m_feature_threshold;
  int m_rec_threshold; // this should be set somewhere (Ara)
        int m_str1_len;
        int m_str2_len;

  int m_diff;
  float m_float_diff;

  int m_rec_diff;
  float m_rec_float_diff;

  // number of differences between the substrings s1[0..i] and s2[0..j]
  int m_diff_array[NAME_SIZE + 1][NAME_SIZE + 1];

  float m_diff_float_array[NAME_SIZE + 1][NAME_SIZE + 1];

  int m_rec_diff_array[NAME_SIZE + 1][NAME_SIZE + 1];

  float m_rec_diff_float_array[NAME_SIZE + 1][NAME_SIZE + 1];

  // feature differences between any two phonemes
public:
  byte m_fd_matrix[256][256];
```

```
  // this will accomodate the new float values for the feature
  // distances in the distance.rul file
  float m_fd_float_matrix[256][256];
public:
  byte *get_fd_matrix();

private:
  // read a file of feature differences
  const bool set_distances(const char *fname);

  const bool set_float_distances(const char *fname);

  // read a file of rec feature differences (by Ara)
  // this is not fully implemented yet
  const bool set_rec_distances(const char *);

        void reset();

  /////////////////////////////////////////////////////////////////
  // plain implementation of the edit distance algorithm
  // this version does not use feature distance tables for
  // anything.
  //
public:
  const int plain_edit_distance(const unsigned char *,
                                const unsigned char *,
                                float &score);

  const int plain_diff(const unsigned char *,
                       const unsigned char *);

  float get_plain_score();

  // this does not look into any feature distance tables
  inline bool plain_char_eq(const byte ch1, const byte ch2)
  {
    // addapted function (Ara)
    if (ch1 == ch2)
      return true;
    else
      return false;
  }

  int get_plain_diff()
  {
    return m_diff;
  }
private:
  //
  /////////////////////////////////////////////////////////////////


  /////////////////////////////////////////////////////////////////
  // original implementation
  // these functions handle character comparison using
  // feature distances to see if its above or below a threshold
  // calculate a score based on difference and string lengths

  double get_score()
        {
              return  1.0 - m_diff / static_cast<double>(__max(m_str1_len, m_str2_len));
        }

  // get the number of differences between the two strings
        int get_difference()
        {
                return m_diff_array[m_str1_len][m_str2_len];
        }

   // the features distance file has multiplied all values by 10
   inline bool char_eq(const byte ch1, const byte ch2)
   {
     // addapted function (Ara)
     if (ch1 == ch2) // if ch1 == ch2 no need to look in matrix
       return true;  // just return true
     else            // else check matrix for threshold value
       return m_fd_matrix[ch1][ch2] <= m_feature_threshold * 10;
   }

 public:
   int get_diff()
   {
     return m_diff;
   }
 private:
   //
   ////////////////////////////////////////////////////////////

   ////////////////////////////////////////////////////////////
   // value using implementation
   // these functions handle character comparison using
   // the feature values as the actual vales returned

   //double get_float_score()
   //{
   //  return 1.0 - m_float_diff / static_cast<double>(__max(m_str1_len, m_str2_len));
   //}


   float get_float_score()
   {
     return 1.0 - m_float_diff / static_cast<float>(__max(m_str1_len, m_str2_len));
   }

   float get_rec_gen_float_score()
   {
     return m_float_diff / static_cast<float>(__max(m_str1_len, m_str2_len));
   }

   float get_float_difference()
         {
                 return m_diff_float_array[m_str1_len][m_str2_len];
         }

   inline float char_float_eq(unsigned int ch1, unsigned int ch2)
   {
     if (ch1 < 0 || ch1 > 255)
       AfxMessageBox("Problem with ch1");

     if (ch2 < 0 || ch2 > 255)
       AfxMessageBox("Problem with ch2");

     if (ch1 == ch2)
       return 0.0;
     else
     {
       //return (static_cast<float>(m_fd_matrix[ch1][ch2]) * (100.0/150.0)) / 100.0;
       return m_fd_float_matrix[ch1][ch2];
     }
   }
```

```
 public:
   float get_float_diff()
   {
     return m_float_diff;
   }
 private:
   //
   ////////////////////////////////////////////////////////////

   ////////////////////////////////////////////////////////////
   // REC implementation where the values in the matrix
   // are used to see if its above or below a threshold
   //
   double get_rec_score()
   {
     return 1.0 - m_rec_diff / static_cast<float>(__max(m_str1_len, m_str2_len));
   }

         int get_rec_difference()
         {
                 return m_rec_diff_array[m_str1_len][m_str2_len];
         }

   // the rec distance is values from 0 to 100 there is no need
   // yet to multiply them by 10 (Ara)
   inline bool rec_eq(const unsigned char r1,
                      const unsigned char r2,
                      float diff_matrix[256][256])
   {
     // addapted function (Ara)
     if (r1 == r2) // if r1 == r2 then there is no need to look in the matrix
       return true;
     else
       return diff_matrix[r1][r2] >= m_rec_threshold;
   }

 public:
   int get_rec_diff()
   {
     return m_rec_diff;
   }
 private:
   //
   ////////////////////////////////////////////////////////////

   ////////////////////////////////////////////////////////////
   // REC implementation where the values are returned to
   // calculate frational differences
   //
   double get_rec_float_score()
   {
     return 1.0 - m_rec_float_diff / static_cast<float>(__max(m_str1_len, m_str2_len));
   }

   float get_rec_float_difference()
   {
     return m_rec_diff_float_array[m_str1_len][m_str2_len];
   }

   inline float rec_float_eq(const unsigned char r1,
                             const unsigned char r2,
                             float diff_matrix[256][256])
   {
     if (r1 == r2)
       return 0;
     else
```

```
    return diff_matrix[r1][r2];
  }

public:
  float get_rec_float_diff()
  {
    return m_rec_float_diff;
  }
private:
  //
  ////////////////////////////////////////////////////////////



  // one of these will be selected based on the desired maximum feature-
  // distance threshold--exact does a quick string compare instead of a
  // more expensive calculation
  const int approx_diff(const unsigned char *str1,
                        const unsigned char *str2);

  // this version actually uses the values stored in the
  // m_fd_matrix instead of just using them to check against
  // a threshold. (Ara)
  const float approx_float_diff(const unsigned char *str1,
                                const unsigned char *str2);

  // work on arrays of Reg-exp codes. (by Ara)
  /*
  const int approx_rec_diff(const unsigned char *arr1,
                            const unsigned char *arr2,
                            const unsigned char diff_matrix[256][256]);


  */

  const int approx_rec_diff(const unsigned char *arr1,
                            const unsigned char *arr2,
                            float diff_matrix[256][256]);

  // the version that uses floats
  /*
  const float approx_rec_float_diff(const unsigned char *arr1,
                                    const unsigned char *arr2,
                                    const unsigned char diff_matrix[256][256]);


  */
  const float approx_rec_float_diff(const unsigned char *arr1,
                                    const unsigned char *arr2,
                                    float diff_matrix[256][256]);



  const int exact_diff(const unsigned char *str1,
                       const unsigned char *str2);

public:
      CApprox(const char *fname1, const char *fname2)
  {
    m_feature_threshold = 1;
    m_rec_threshold = 50; // this should be set somewhere
        for (int p = 0; p <= NAME_SIZE; p++)
    {
      // this loads the int type array
              m_diff_array[p][0] = p;
```

```
              m_diff_array[0][p] = p;

      // this load the float array
      m_diff_float_array[p][0] = p;
      m_diff_float_array[0][p] = p;

      m_rec_diff_array[p][0] = p;
      m_rec_diff_array[0][p] = p;

      m_rec_diff_float_array[p][0] = p;
      m_rec_diff_float_array[0][p] = p;
        }
    // loads the 256 X 256 array from the difference file
    // this is only used by the character distance algorithm
    // file originaly written by Robert Drabek

    // fname1 should be the original distance.rul file
    m_good = set_distances(fname1);

    // here is the new function that loads the float values
    // into a float array
    // fname2 should be the new distance file called
    // floatdist.rul
    m_good = set_float_distances(fname2);

    m_str1_len = m_str2_len = 0;
    m_diff = -1;
    m_float_diff = -1.0;
  }

  // this version of the constructor is only used when
  // generating the REC distance file
      CApprox(const char *fname1)
  {
    m_feature_threshold = 1,
    m_rec_threshold = 50; // this should be set somewhere
        for (int p = 0; p <= NAME_SIZE; p++)
    {
      m_diff_float_array[p][0] = p;
      m_diff_float_array[0][p] = p;
        }
    m_good = set_float_distances(fname1);
    m_str1_len = m_str2_len = 0;
    m_diff = -1;
    m_float_diff = -1.0;
  }


  const int differences(const unsigned char *str1,
                        const unsigned char *str2,
                        const int threshold,
                        double &score);

  const float float_differences(const unsigned char *str1,
                                const unsigned char *str2,
                                float &score);

  const float rec_gen_float_differences(const unsigned char *str1,
                                        const unsigned char *str2,
                                        float &score);
  /*
      const int rec_differences(const unsigned char *recArray1,
                                const unsigned char *recArray2,
                                const unsigned char recCompArray[256][256],
                                double &score);
        const float rec_float_differences(const unsigned char *recArray1,
                                          const unsigned char *recArray2,
                                          const unsigned char recCompArray[256][256],
                                          float &score);
    */

        const int rec_differences(const unsigned char *recArray1,
                                  const unsigned char *recArray2,
                                  float recCompArray[256][256],
                                  double &score);

        const float rec_float_differences(const unsigned char *recArray1,
                                          const unsigned char *recArray2,
                                          float recCompArray[256][256],
                                          float &score);


    const bool good() const
    {
      return m_good;
    }

    void set_fd_threshold(const int t)
    {
      m_feature_threshold = t;
    }

    void set_rec_threshold(const int t)
    {
      m_rec_threshold = t;
    }

    const int get_fd_threshold()
    {
      return m_feature_threshold;
    }

    const int get_rec_threshold()
    {
      return m_rec_threshold;
    }


  };

  #endif

  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
/*

        Function to transform a UCV string (a string of IPA consonants)
        into a 32 bit value.

        The value is created by walking through the string, and looking
        at each character separately.  Each character is mapped to a
        particular bit within the 32 bit value.

        Note that this method of encoding is limited because we loose
        information about the number of occurances of each character,
        since we only get one bit for each character.
*/

#ifndef BITMAPIZE_UCV_DEFFED
#define BITMAPIZE_UCV_DEFFED

typedef struct  bits_64_tag
{
        unsigned int    int1;
        unsigned int    int2;

}       bits_64_t;

bool operator<(const bits_64_t &b1, const bits_64_t &b2);
bool operator==(const bits_64_t &b1, const bits_64_t &b2);

void                                            initConsonantBitvalArray();
//      unsigned        int             bitmapizeUCV(const char *ucv);
void    bitmapizeUCV(const char *ucv, unsigned int *bitSignature, int numBytesInSignature);


#endif
```

```
/*
* cgrep.h --- regular expression searching using the shortest substring model

  Copyright (C) 1995 Charles L. A. Clarke.  All rights reserved.

  Distribution of this software and its documentation is subject to the
  following terms and conditions:

  1. The software or its documentation may not be sold or exchanged for profit.

  2. The software or its documentation may not be included in any software,
     device, or process which is sold, exchanged for profit, or for which
     a licence or royalty fee is charged.

  3. Permission is granted to use this software and its documentation for
     purposes not covered by the above conditions (including for
     educational, research, or commercial purposes, and including
     modification and redistribution), provided that the copyright notice,
     this permission notice, and the following disclaimer is included and
     appear in all supporting documentation.

  NO WARRANTY

  BECAUSE THE SOFTWARE IS PROVIDED FREE OF CHARGE THERE IS NO WARRANTY,
  TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED
  IN WRITING THIS SOFTWARE IS PROVIDED BY ITS AUTHOR AND/OR OTHER PARTIES
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
  PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT UNLESS REQUIRED BY
  APPLICABLE LAW OR AGREED TO IN WRITING SHALL THE AUTHOR OR ANY OTHER
  PARTY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
  OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
  THE POSSIBILITY OF SUCH DAMAGE.

*/


int             RexParse(unsigned char* expr);
int             RexGetState( unsigned *st1, unsigned *st2, unsigned *sy);
void    RexCleanup();
```

```
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'p'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'b'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'f'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'v'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'w'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'p'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'b'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'f'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'v'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'w'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'p'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'b'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'f'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'v'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'w'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'p'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'b'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'f'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'v'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'w'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'p'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'b'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'f'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'v'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'w'].int1 = 1;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'m'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'n'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ɛ'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ɛ'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'m'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'n'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ɛ'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ɛ'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'m'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'n'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'n'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ɛ'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ɛ'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'m'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'n'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'ɛ'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'ɛ'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'m'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'n'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ɛ'].int1 = 2;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ɛ'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'t'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'d'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'»'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ê'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'t'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'d'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'»'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ê'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'t'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'d'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'»'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ê'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'t'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'d'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'»'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'ê'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'é'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'t'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'d'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'»'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ê'].int1 = 4;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'s'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'z'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'"'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'"'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'{'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'{'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)';'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ê'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'ê'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'j'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'s'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'z'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'*'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'*'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'¶'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)';'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ê'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'j'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'s'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'z'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'*'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'*'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'¶'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)';'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ê'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'j'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'s'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'z'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'*'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'*'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'¶'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)';'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'ê'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'j'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'s'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'z'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'*'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'*'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'¶'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)';'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ê'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'j'].int1 = 8;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'k'].int1 = 16;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'g'].int1 = 16;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'h'].int1 = 16;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'k'].int1 = 16;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'k'].int2 = 0;
```

```
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'g'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'g'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'h'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'h'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'k'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'k'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'g'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'g'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'h'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'h'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'k'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'k'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'g'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'g'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'h'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'h'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'k'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'k'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'g'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'g'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'h'].int1 = 16;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'h'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'l'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'l'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'r'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)'r'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'l'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'l'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'r'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)'r'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'l'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'l'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'r'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)'r'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'l'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'l'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'r'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)'r'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'l'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'l'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'r'].int1 = 32;
|     consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)'r'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'p'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'p'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'b'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'b'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'f'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'f'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'v'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'v'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'w'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'w'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'p'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'p'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'b'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'b'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'f'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'f'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'v'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'v'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'w'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'w'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)'p'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)'p'].int2 = 0;
|     consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)'b'].int1 = 64;
|     consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)'b'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'f'].int1 = 64;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'v'].int1 = 64;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'w'].int1 = 64;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'m'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'n'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'£'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'m'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'n'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'£'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'m'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'n'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'£'].int1 = 128;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'t'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'d'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'•'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'ê'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'t'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'d'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'•'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'ê'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'t'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'d'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'•'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'ê'].int1 = 256;
consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'s'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'z'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'"'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'"'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'{'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'{'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)';'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'ê'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'j'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'s'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'z'].int1 = 512;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'z'].int2 = 0;
```

```
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'"'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'"'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'{'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'{'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)';'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)';'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'ê'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'ê'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'j'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'j'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'s'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'s'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'z'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'z'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£"][(unsigned char)'"'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'"'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'{'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'{'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)';'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)';'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'ê'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'ê'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'j'].int1 = 512;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'j'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'k'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'k'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'g'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'g'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'h'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'h'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'k'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'k'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'g'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'g'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'h'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'h'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'k'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'k'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'g'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'g'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'h'].int1 = 1024;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'h'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'l'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'l'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'r'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)'r'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'l'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'l'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'r'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)'r'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'l'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'l'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'r'].int1 = 2048;
|  consonantpairBitvalArray[(unsigned char)'£'][(unsigned char)'r'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'p'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'p'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'b'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'b'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'f'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'f'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'v'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'v'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'w'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'w'].int2 = 0;
|  consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'p'].int1 = 4096;
|  consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'p'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'b'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'f'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'v'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'w'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'p'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'b'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'f'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'v'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'w'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'p'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'b'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'f'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'v'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'w'].int1 = 4096;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'m'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'n'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'£'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'m'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'n'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'£'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'m'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'n'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'£'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'m'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'n'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'£'].int1 = 8192;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'t'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'d'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'»'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'ê'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'t'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'d'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'»'].int1 = 16384;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'»'].int2 = 0;
```

```
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'ê'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'t'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'t'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'d'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'d'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'»'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'»'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'ê'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'t'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'t'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'d'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'d'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'»'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'»'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int1 = 16384;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'s'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'s'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'z'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'z'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'"'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'"'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'¶'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'¶'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)';'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)';'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'ê'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'j'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'j'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'s'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'s'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'z'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'z'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'"'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'"'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'¶'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'¶'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)';'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)';'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'ê'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'j'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'j'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'s'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'s'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'z'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'z'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'"'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'"'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'¶'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'¶'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)';'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)';'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'ê'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'ê'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'j'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'j'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'s'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'s'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'z'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'z'].int2 = 0;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'"'].int1 = 32768;
| consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'"'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'§'].int1 = 32768;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'§'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'¡'].int1 = 32768;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'¡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int1 = 32768;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'j'].int1 = 32768;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'k'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'g'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'h'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'k'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray](unsigned char)'d'][(unsigned char)'g'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'h'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'k'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'g'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'h'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'k'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'g'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'h'].int1 = 65536;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'l'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'r'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'l'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'r'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'l'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'r'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'»'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'l'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'e'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'é'][(unsigned char)'r'].int1 = 131072;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'v'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'§'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)';'](unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'p'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'b'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'f'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'v'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'w'].int1 = 262144;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'E'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'E'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'E'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'E'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)'m'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'£'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'£'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'£'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'£'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'m'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'n'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'£'].int1 = 524288;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'è'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'è'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'»'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)';'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'t'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'d'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'»'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'ê'].int1 = 1048576;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'¶'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)';'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'¶'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)';'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'¶'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'¶'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)';'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)';'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¶'][(unsigned char)'•'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'‡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'‡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'¡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'¡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'‡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'‡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'¡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'¡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'‡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'‡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'¡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'¡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'s'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'z'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'•'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'•'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'‡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'‡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'¡'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'¡'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'ê'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'j'].int1 = 2097152;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'j'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'g'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'k'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'k'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'g'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'g'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'h'].int1 = 4194304;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'h'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'l'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'l'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'r'].int1 = 8388608;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)'r'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'p'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'b'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'f'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'v'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'w'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'p'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'p'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'b'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'f'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'v'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'w'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'p'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'b'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'f'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'v'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'w'].int1 = 16777216;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'m'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'n'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'£'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'m'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'n'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'£'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'m'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'m'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'n'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'n'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'£'].int1 = 33554432;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'£'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'t'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'d'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'»'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'ê'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'t'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'d'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'»'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'ê'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'t'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'t'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'d'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'d'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'»'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'»'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'ê'].int1 = 67108864;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'ê'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'s'].int1 = 134217728;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'s'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'z'].int1 = 134217728;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'z'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'"'].int1 = 134217728;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'"'].int2 = 0;
```

```
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'§'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'§'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)';'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)';'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'ê'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'ê'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'j'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'j'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'s'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'s'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'z'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'z'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'"'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'"'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'§'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'§'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)';'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)';'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'ê'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'ê'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'j'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'j'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'s'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'s'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'z'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'z'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'"'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'"'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'§'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'§'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)';'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)';'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'ê'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'ê'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'j'].int1 = 134217728;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'j'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'k'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'k'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'g'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'g'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'h'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'h'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'k'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'k'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'g'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'g'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'h'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'h'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'k'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'k'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'g'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'g'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'h'].int1 = 268435456;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'h'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'l'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'l'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'r'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)'r'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'l'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'l'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'r'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)'r'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'l'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'l'].int2 = 0;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'r'].int1 = 536870912;
|    consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)'r'].int2 = 0;
```

```
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'b'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'f'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'v'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'w'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'p'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'p'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'b'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'b'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'f'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'f'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'v'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'v'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'w'].int1 = 1073741824;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'w'].int2 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'m'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'m'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'n'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'n'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ɛ'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ɛ'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'m'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'m'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'n'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'n'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ɛ'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ɛ'].int2 = 2147483648;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'t'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'t'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'d'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'d'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'•'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'•'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ê'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ê'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'t'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'t'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'d'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'d'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'•'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'•'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ê'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ê'].int2 = 1;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'s'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'s'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'z'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'z'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'"'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'"'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'§'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'§'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'¡'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'¡'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ê'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'ê'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'j'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'j'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'s'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'s'].int2 = 2;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'z'].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'z'].int2 = 2;
```

```
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'"'].int2 = 2;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'§'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'§'].int2 = 2;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'¡'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'¡'].int2 = 2;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ê'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'ê'].int2 = 2;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'j'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'j'].int2 = 2;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'k'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'k'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'g'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'g'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'h'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'h'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'k'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'k'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'g'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'g'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'h'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'h'].int2 = 4;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'l'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'l'].int2 = 8;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'r'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)'r'].int2 = 8;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'l'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'l'].int2 = 8;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'r'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)'r'].int2 = 8;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'p'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'p'].int2 = 16;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'b'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'b'].int2 = 16;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'f'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'f'].int2 = 16;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'v'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'v'].int2 = 16;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'w'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'w'].int2 = 16;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'m'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'m'].int2 = 32;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'n'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'n'].int2 = 32;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ɛ'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ɛ'].int2 = 32;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'t'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'t'].int2 = 64;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'d'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'d'].int2 = 64;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'•'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'•'].int2 = 64;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ê'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ê'].int2 = 64;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'s'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'s'].int2 = 128;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'z'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'z'].int2 = 128;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'"'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'"'].int2 = 128;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'§'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'§'].int2 = 128;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'|'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'¡'].int2 = 128;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ê'].int1 = 0;
|   consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'ê'].int2 = 128;
```

```
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'j'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'j'].int2 = 128;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'k'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'k'].int2 = 256;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'g'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'g'].int2 = 256;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'h'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'h'].int2 = 256;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'l'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'l'].int2 = 512;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'r'].int1 = 0;
consonantpairBitvalArray[(unsigned char)' '][(unsigned char)'r'].int2 = 512;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'p'][(unsigned char)' '].int2 = 1024;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'b'][(unsigned char)' '].int2 = 1024;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'f'][(unsigned char)' '].int2 = 1024;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'v'][(unsigned char)' '].int2 = 1024;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'w'][(unsigned char)' '].int2 = 1024;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'m'][(unsigned char)' '].int2 = 2048;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'n'][(unsigned char)' '].int2 = 2048;
consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'E'][(unsigned char)' '].int2 = 2048;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'t'][(unsigned char)' '].int2 = 4096;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'d'][(unsigned char)' '].int2 = 4096;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'•'][(unsigned char)' '].int2 = 4096;
consonantpairBitvalArray[(unsigned char)'è'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'è'][(unsigned char)' '].int2 = 4096;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'s'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'z'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'"'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'‡'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'¡'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'ê'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'j'][(unsigned char)' '].int2 = 8192;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'k'][(unsigned char)' '].int2 = 16384;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'g'][(unsigned char)' '].int2 = 16384;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'h'][(unsigned char)' '].int2 = 16384;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'l'][(unsigned char)' '].int2 = 32768;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)' '].int1 = 0;
consonantpairBitvalArray[(unsigned char)'r'][(unsigned char)' '].int2 = 32768;
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _DISTANCE_H_
#define _DISTANCE_H_


#include "stdafx.h"
#include <fstream>

using namespace std;

extern ofstream err;

extern int aDistance[255][255];

class CDistance
{
        int m_SymbolThreshold;
public:

        CDistance() { SetDefaults(); }

        void SetDefaults();

        void SetDistance(UCHAR c1, UCHAR c2, int dist);
        void SetThreshold(int thresh)
        {
                m_SymbolThreshold = thresh;
        }
        int GetDistance( UCHAR c1, UCHAR c2 )
        {
                return aDistance[c1][c2];
        }
        BOOL Pass(UCHAR c1, UCHAR c2)
        {
                return ( aDistance[c1][c2] <= m_SymbolThreshold );
        }

        BOOL SetDistance( LPCTSTR line );

        BOOL Reader(LPCTSTR infile = NULL);

};


#endif // #ifdef _DISTANCE_H
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// editDistance.h: interface for the CEditDistance class.
//
//////////////////////////////////////////////////////////////////////

#ifndef _EDITDIST_H
#define _EDITDIST_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef EOS
#define EOS '\0'
#endif

#ifndef STRINGMAX
#define STRINGMAX 30
#endif

//maximum score for one edit - to get a score between 0 and 1, divide
//the number of edits by this number
#ifndef DIFFMAX
#define DIFFMAX 10000
#endif

#include "simpcodedisttable.h"

class CEditDistance
{
private:
        int                                     distanceArray[STRINGMAX+1][STRINGMAX+1];
        float                           threshold;
        int                                     thresholdInt;
        CSimpCodeDistTable      *simpCodedistTable;

public:
        CEditDistance(CSimpCodeDistTable *aSimpCodedistTable);

        virtual ~CEditDistance();

        void setThreshold(float inThreshold)
        {
                threshold    = inThreshold;
                thresholdInt = (threshold * DIFFMAX);
        }

        float getThreshold()    {       return(threshold);}

        int getDistanceArrayCell(int, int);

        float getDistanceScore(unsigned char*,unsigned char*);
        float getDistanceScoreWithEarlyOut(unsigned char *str1,

                                                unsigned char *str2);

};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   ExtRName.h: interface for the ExtRNameClass.

   Winter 1998: Ed Barker - first version.


   1. General.

   This file defines the Ranker ExtRNameClass, which is a subclass
   of RName.  It was created so that we coul tack extra data onto the
        RName objects we pass to the ranker, and then retrieve the data
        when the ranker gives us back the name objects.
   certain criteria.
*/

#if !defined(_EXT_RNAME_H)
#define _EXT_RNAME_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000



#include        "ranker.h"
#include        "TDS_CONSTANTS.h"
#include        "tdssearcher_enums.h"


//      we should not need this, but for some reason the header
//      file for some of the template stuff requires that NameRecord
//      be defined (even though it only should be looking at a pointer
//      to NameRecord, and should be able to get by with a forward
//      class declaration.
//#include      "NameRecord.h"

class ExtRName : public RName {
private:

        char                                            nameCode[TDS_MAX_NAME_CODE + 1];

  void assign(const ExtRName &c);

public:
  // explicit default constructor
        ExtRName();

        // custom constructor
        ExtRName(const string &n, const float ps, const byte c[],
                           const char v, const bool e, char *aNameCode,
                                         e_tds_culture aClassCulture,
                                         e_tds_culture aPipeCulture);

        // copy constructor
        ExtRName(const ExtRName &c) { assign(c); }

        // assignment operator
        const ExtRName& operator=(const ExtRName &c) { assign(c); return c; }

        char                                            *getNameCode()  { return nameCode; }
};
```

```
#endif // !defined(_EXT_RNAME_H)

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// feature distance object header file

/*
  The CFeatureDistanceTable class encapsulates the feature
  distance table.  The feature distance table is composed of
  the IPA characters and their feature distances stored as float
  values.

  m_feature_distance_table contains the values
  m_status is a generic status variable that can be checked
  with the get_status() function.  If m_status is false then
  there is a problem with the table.  Usually this will be due
  to the fdist.dat file not being loaded in properly.

  the overloaded function get_feature_distance can take either
  two ints or two unsigned chars as the arguments and will return
  a float value that is the feature distance between those two
  IPA characters.
*/

#ifndef _FEATURE_DISTANCE_TABLE_H
#define _FEATURE_DISTANCE_TABLE_H

class CFeatureDistanceTable
{
// data members
private:
  float m_feature_distance_table[256][256];
  bool m_status;


// functions
private:
  bool load_array(const char *);
  void dump();

public:
  CFeatureDistanceTable() { m_status = false; }
  CFeatureDistanceTable(const char *);
  ~CFeatureDistanceTable();
  bool get_status() { return m_status; }
  float get_feature_distance(unsigned char chx, unsigned char chy)
  {
    return m_feature_distance_table[chx][chy];
  }
};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// FeatureEditDistance h: interface for the FeatureEditDistance class.
//
//////////////////////////////////////////////////////////////////////

#ifndef _FEATURE_EDITDIST_H
#define _FEATURE_EDITDIST_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef EOS
#define EOS '\0'
#endif

#ifndef STRINGMAX
#define STRINGMAX 30
#endif

//maximum score for one edit - to get a score between 0 and 1, divide
//the number of edits by this number
#ifndef DIFFMAX
#define DIFFMAX 10000
#endif

#include "featuredistancetable.h"

class FeatureEditDistance
{
private:
        float                                                   distanceArray[STRINGMAX+1][STR
 »» INGMAX+1];
        float                                                   threshold,
        CFeatureDistanceTable   *featureDistTable;

public:
        FeatureEditDistance(CFeatureDistanceTable *aFeatureDistTable);

        virtual ~FeatureEditDistance();

        void setThreshold(float inThreshold)
        {
                threshold = inThreshold;
        }

        float getThreshold()    {       return(threshold);}

        float getDistanceArrayCell(int i, int j)
        {
                if (i > STRINGMAX+1)
                        return(-1);
                if (j > STRINGMAX+1)
                        return(-2);
                return(distanceArray[i][j]);
        }

        float getDistanceScore(unsigned char*,unsigned char*);
        float getDistanceScoreWithEarlyOut(unsigned char *str1,
 »»                                     unsigned char *str2);

};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _GROUPDATA_H_
#define _GROUPDATA_H_


#include        "SimpEditDist.h"

#include        "GroupNameOffsetVectors.h"

#pragma warning(disable: 4786)

#include        <use_ansi.h>
#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>
#include        <cstring>
#include  "approx.h"
//      #include <stl.h>

using namespace std;

#define         TDS_MAX_GROUP_SIZE              30

//      save the current pack value
#pragma pack( push, save_pack )

#pragma pack(1)

//      the structure below describes and entry in the GroupData map.
typedef struct group_map_entry_tag
{
        char                            group[TDS_MAX_GROUP_SIZE + 1];
  unsigned int  nameOffsetsOffset;
  unsigned int  numNameOffsets;
} group_map_entry_t;

typedef group_map_entry_t       *group_map_entry_ptr_t;


// This file contains the definitions for the GroupData Class
class GroupData
{
public:

        GroupData(const char *fileName, const char *groupNameOffsetsFileName,
                                CSimpleEditDistance *anEditDistancizer,
                                float anEditDistThresh);
        ~GroupData();

        bool    getStatus()     {return status;}

        //      function to do a binary search on the groups we have loaded, looking
        //      the groups specified in groupSet.  The function returns a set
        //      of unsigned ints, which are offsets into the names file of all
        //      the names that might be exact phonetic matches (since at least
        //      one of the groups they generate matches one of the groups the
        //      query genereated exactly.
        set<unsigned int>       *       getNameOffsetsOfExactGroupMatches(set<CString> &groupSet);

        //      function to compare a set of group strings to all the groups in
        //      the group file we have loaded into memory.  For each group comparison,
```

```
        //      we perform an edit distance comparison.  If the resulting score
        //      beats the threshold (set via the constructor), we collect the
        //      offsets (of names) associated with the group.
        set<unsigned int>       *       getNameOffsetsForGroups(set<CString> &groupSet,

  =>                                                                    int *numGroupsThatPassedGroupEd
  => itDist,

  =>                                                                    bool    *stopVar);

        unsigned int    getNumGroups()  {return numGroupsInMap;}

        void                            setEditDistThresh(float anEditDistThresh) {editDistThresh = an
  => EditDistThresh;}


private:
        group_map_entry_t                       *mapData;
        unsigned int                                    numGroupsInMap;
        bool                                                            status;
        CSimpleEditDistance             *editDistancizer;
        float                                                           editDistThresh;
        GroupNameOffsetVectors  *groupNameOffsetVectors;

};


//      restore previous pack value
#pragma pack( pop, save_pack )


#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _GROUPNAMEOFFSETVECTORS_H
#define _GROUPNAMEOFFSETVECTORS_H


#pragma warning(disable: 4786)

#include        <use_ansi.h>
#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>
#include        <cstring>
#include  "approx.h"
//      #include <stl.h>

using namespace std;



// This file contains the definitions for the GroupNameOffsetVectors Class
//      which is a class abstract the file of name offsets that is associated with
//      a group index file.
class GroupNameOffsetVectors
{
public:

        GroupNameOffsetVectors(const char *fileName, bool useMemory);
        ~GroupNameOffsetVectors();

        bool    getStatus()     {return status;}

        //      function to do a binary search on the groups we have loaded, looking
        //      the groups specified in groupSet.  The function returns a set
        //      of unsigned ints, which are offsets into the names file of all
        //      the names that might be exact phonetic matches (since at least
        //      one of the groups they generate matches one of the groups the
        //      query genereated exactly.
        bool    addNameOffsets(set<unsigned int> *offsetSet,
                                                                                unsigned int s
  => tartOffset,
                                                                                unsigned int n
  => umOffsets);

private:
        FILE                                                            *groupNameOffsetsFile;
  =>  //      file containing the offsets (into the names file)
        unsigned int                                    *groupNameOffsets;
        unsigned int                                    totalOffsets;
        unsigned int                                    lastOffset;
        bool                                                            status;
        bool                                                            usingMemory;
        CString                                                 groupNameOffsetsFileName;

};



#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _LASNFA_H
#define _LASNFA_H

#include "stdafx.h"
#include <afxtempl.h>
#include <iostream>

using namespace std;


// The final state in a transition table is defined as NOT zero
// this works out to be -1 (a -1 would work too) but states are
// unsigned.
//
#ifndef FINAL
#define FINAL  ~(unsigned)0
#endif

// Some special symbols used in regular expressions. Not currently
// supporting the START and STOP symbols since all our processing is
// on strings and we are manually inserting spaces as a word delimeter
// if they are not already there.
//
#define SYMBOLS 256
#define SYMBOL_START (SYMBOLS + 0)
#define SYMBOL_END (SYMBOLS + 1)
#define SYMBOL_START_LINE (SYMBOLS + 2)
#define SYMBOL_END_LINE (SYMBOLS + 3)
#define SYMBOL_SIGMA (SYMBOLS + 4)  /* Highest numbered valid superSymbol. */
#define SYMBOL_LAMBDA (SYMBOLS + 5)
#define SYMBOL_INVALID (SYMBOLS + 6)

// References to some C (non C++) routines from cgrep.c and stored as
// object code in cgrepdll.dll.  These routines parse the regular
// expressions and return state information for building transition
// tables.
//
/*
extern "C"
{
        int RexParse(unsigned char* expr);
        int RexGetState( unsigned *st1, unsigned *st2, unsigned *sy);
}
*/


// CTransition is the building block for a state table. Little
// more than a structure it describes a transition from the current
// state to the next state.  The current state is defined by this class
//
class CTransition
{
private:
        void assign(const CTransition &t) { next=t.next; symbol=t.symbol;}

public:
        TCHAR next;
        TCHAR symbol;
public:
        CTransition(const CTransition& t) { assign(t); }
        CTransition() { next=(TCHAR)FINAL; symbol=(TCHAR)0; }
        CTransition( TCHAR n, TCHAR s) { Set(n,s); }
        void Set( TCHAR n, TCHAR s) { next=n; symbol=s; }
        const CTransition& operator=(const CTransition &t) { assign(t); return t; }
```

```
};

// CState is nothing more than an array of CTransition objects.
// This describes a state as a list of all the ways of going on to
// the next state. Since the array m_aTrans can be any size, we have
// multi-way branching in our state tables. Not just binary branching.
// An object of type CState will be stored in a state table. The ID of
// any CState object is defined by its place in the state table. The
// first object added is state 0 etc.
class CState
{
public:
        CArray<CTransition,CTransition> m_aTrans;
        int GetSize(){return m_aTrans.GetSize();}
        int Add( TCHAR next, TCHAR symbol )
        {
                CTransition trans(next,symbol);
                return m_aTrans.Add(trans);
        }
};

// This is the actual state table. Basicly it's an array of CState
// objects and a bunch of routines to manipulate them, search text,
// compare with others, etc.  In some cases two different methods
// were used where one general one might make it more clear. This was
// done for run time optimization.
//
// There is more work to be done on this class, Many of the public data
// and methods could be private. Much of the further development in
// similarity is taking place here.
//
class CNfa
{
public:
        BOOL InsertNfa(int start, int stop, CNfa& nfa);
        CNfa(){m_bCancel = FALSE; m_bSigma = FALSE; }

        BOOL m_bCancel;

        BOOL BufMatch(LPCTSTR buf, int gcount);
        BOOL BufMatchWalk(LPCTSTR, LPCTSTR, TCHAR, TCHAR, int &);
        BOOL SubBufMatch(LPCTSTR buf, int gcount);
        BOOL SubBufMatchWalk(LPCTSTR, LPCTSTR, TCHAR, TCHAR);

        CString m_SourceName;
        CString m_Rex;
        CString m_FirstChars;
        CString m_LastChars;

        BOOL m_bBoundry;
        BOOL m_bLambda;
        BOOL m_bSigma;
        BOOL m_bReverse;

        CArray<CState,CState> m_states;
        CArray<CString,CString> m_variants;

        int Add(TCHAR state, TCHAR next, unsigned symbol );
        BOOL Set(LPCTSTR pRex, BOOL bReverse=FALSE);
        void trace(ostream&);

        void AddToFirstChars(TCHAR);
        void AddToLastChars(TCHAR);
        BOOL IsEmpty(){ return ( m_states.GetSize() <= 0);}

        void RemoveAll(){ m_states.RemoveAll(); m_FirstChars.Empty(); m_variants.RemoveAll();}
```

```
	int GetSize(){return m_states.GetSize();}

	void Walk(TCHAR state, CString str, char chr = NULL);

	BOOL Match(LPCTSTR name, int*);
	BOOL StringMatchWalk(TCHAR state, LPCTSTR name, int* );

	BOOL CompleteMatch(LPCTSTR name, int*);
	BOOL CompleteStringMatchWalk(TCHAR state, LPCTSTR name, int* );

	BOOL LeftMatch(LPCTSTR name, int, int*);
	BOOL LeftStringMatchWalk(unsigned int state, LPCTSTR name, int* );

	BOOL Match(CNfa& nfa);
	BOOL NfaMatchWalk(CNfa&, TCHAR, TCHAR, int=0 );

	BOOL SubMatch(LPCTSTR name, int *pPos, int *pCount);
	BOOL SubStringMatchWalk(TCHAR state, LPCTSTR name, int *pPos, int *pCount);

	int	LightStateTableTraversal(TCHAR state, char chr = NULL);
	int	GetNumVariants();

private:
	int m_MaxState;

};

// The following global friend operators are used to translate
// an in-memory CNfa object to a file or buffer stream.
//
ofstream& operator<< (ofstream& stream, CTransition& t);
ofstream& operator<< (ofstream& stream, CState& st);
ostream&  operator<< (ostream& stream, CNfa& nfa);

#endif  // #ifndef _LASNFA_H
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
//      Class to hold the information about a name (the stuff we need to
//      pass to the ranker) for up to two cultures.
//
//      This class was created so that we could use the STL map to
//      represent a name that is returned from a query in possibly
//      two cultures.  The Ranker requires that if the same name comes
//      back from two cultures, an ExtRName object for both cultures can
//      be submitted, but they must be submitted at the same time.
//
//      Because we have separate files for each culture, each name is assigned
//      a unique id  We use this as a key of uniqueness within the STL
//      map of NameRecord objects.  When we search using our first culture,
//      we create NameRecord objects by reading the data out of that culture's
//      associated names file.  In that file, we find the name, the unique id,
//      the leading consonants, and a leading vowel flag.

//      For the second culture, each time we read from that culture's associated
//      names file, we look to see if that id is already in the map.  If not,
//      we create a new NameRecord object and add it to the map using the
//      unique id as a key.  If it is already found, we simply add our data
//      (edit distance score, starting consonants, etc) to the exiting NameRecord
//      object.
//
//      This object keeps track of a single name and a single exact match flag.
//      The other information (start cons, start vowel flag, edit dist score) are
//      kept track of on a per-culture basis.  Note though that the
//      start cons, start vowel flag data items will always be the same
//      for any name, but the edit distance can vary (depending on the variant
//      the led us to the name).  For this reason, we sometimes only need to
//      check the edit dist score, rather than copy all the data.
//
//      The object keeps a flag that indicates of the first culture described
//      in the data is ANGLO or not.

#ifndef NAME_RECORD_H_DEFED
#define NAME_RECORD_H_DEFED

#include        <stdlib.h>


#include        "TDS_CONSTANTS.h"
//#include      "TDSDlg.h"
#include        "ranker.h"
#include        "ExtRName.h"
#include        "tdssearcher_enums.h"



class NameRecord {

public:

        NameRecord(char *aName, unsigned char *someStartConsonants, char aStartVowelFlag,
                                                float   anEditDistScore, char anExactMatchFlag,
                                                e_tds_culture aCulture,
                                                char *aNameCode)
        {
                numCultures = 1;
                strncpy(nameString, aName, TDS_MAX_NAME);
                nameString[TDS_MAX_NAME] = '\0';
                strncpy((char *)startConsonants[0], (const char *)someStartConsonants, TDS_MAX_START_CONSONANT
  => S);
                startConsonants[0][TDS_MAX_START_CONSONANTS] = '\0';
```

```
                startConsonants[1][0] = '\0';
                startVowelFlag[0] = aStartVowelFlag;
                editDistScore[0] = anEditDistScore;
                editDistScore[1] = -1.0;        //      indicate that it is not set yet
                exactMatchFlag = anExactMatchFlag;
                strncpy(nameCode, aNameCode, TDS_MAX_NAME_CODE);
                nameCode[TDS_MAX_NAME_CODE] = '\0';
                culture[0] = aCulture,
                culture[1] = TDS_CULT_ANGLO;                    // set to something for initialization
        }


        NameRecord()
        {
        }


        void    assign(const NameRecord& nr)    {
                numCultures = nr.numCultures;
                strcpy(nameString, nr.nameString);
                strcpy(nameCode, nr.nameCode);
                strcpy((char *)startConsonants[0], (const char *)nr.startConsonants[0]);
                strcpy((char *)startConsonants[1], (const char *)nr.startConsonants[1]);
                startVowelFlag[0] = nr.startVowelFlag[0];
                startVowelFlag[1] = nr.startVowelFlag[1];
                editDistScore[0] = nr.editDistScore[0];
                editDistScore[1] = nr.editDistScore[1];
                exactMatchFlag = nr.exactMatchFlag;
                culture[0] = nr.culture[0];
                culture[1] = nr.culture[1];
        }

        void    submitToRanker(Ranker &ranker);

        const NameRecord& operator=(const NameRecord &nr)       {
                assign(nr); return nr;
        }

        int     operator<(const NameRecord &nr) const {return 1;}
        int     operator==(const NameRecord &nr)        const {return 1;}


        //      call this function when a name is returned by the filter
        //      (similar or exact), but the name was already returned.
        //      This happens when the same name is returned via two different
        //      cultures.
        //      Note we do not need to re-copy the name or the nameCode since
        //      it is the same.  Also note that we do not need to pass in
        //      the exact match flag.  This is because we do the exact
        //      lookup for exact matches for all cultures before moving
        //      on to the similar search.  Therefore, if we do add the same
        //      name via a different culture, it will always have the same
        //      value for the exactMatchFlag, because we do not even bother
        //      with names that come back from the similar search if we
        //      also found in with the exact match.
        void    updateName(unsigned char *someStartConsonants, char aStartVowelFlag,
                                                                                                float
 =>     anEditDistScore,
                                                                                                e_tds_
 => culture aCulture)
        {
                //      this is the first time we are seeing the second culture
                //      for this name, so copy the data and set the number of
                //      cultures to 2.
                numCultures = 2;
                strncpy((char *)startConsonants[1], (const char *)someStartConsonants, TDS_MAX_START_CONSONANT
», S);
                startConsonants[1][TDS_MAX_START_CONSONANTS] = '\0';
                startVowelFlag[1] = aStartVowelFlag;
                editDistScore[1] = anEditDistScore;
                culture[1] = aCulture;
        }

        int             getNumCultures()        {return numCultures;}
        char    getExactMatchFlag()     {return exactMatchFlag;}


private:

        unsigned char   numCultures;            //      1 or 2, depending on number of cultures

 »                      //      that brought this name back
        char                                    nameString[TDS_MAX_NAME + 1];
        char                                    nameCode[TDS_MAX_NAME_CODE + 1];

        //      these variable are done in arrays of 2 because we
        //      have to reserve space for up to 2 cultures.
        unsigned char   startConsonants[2][TDS_MAX_START_CONSONANTS + 1];
        char    startVowelFlag[2];              // 'Y', 'N' or 'B'
        float   editDistScore[2];

        //      cultures of the name.  Note that if both are specified, the first
        //      is always anglo.
        e_tds_culture   culture[2];

        char                                                                    exactMatchFlag;
 »                      //      'Y' or 'N'

};


#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _PARSE_H_
#define _PARSE_H_

#ifndef _LASNFA_H
#include "lasnfa.h"
#endif

#include "simpcodedistTable.h"
#include "featuredistancetable.h"

#define         MAX_RULES_FILE_LINE             500
#define         MAX_SYMBOL_LEN                          100
#define         MAX_VALUE_LEN                                   300


#define REPORT_MODE_SHOW_ALL_RULES      1
#define REPORT_MODE_JUST_SHOW_CODES     2


//      an enumerated type to describe problems with
//      variant generation
enum            tds_vargen_code {
        TDS_VARGEN_CODE_OK,                                     //      no problems
        TDS_VARGEN_CODE_BAD_REGEX,              //      regex generated (or supplied) was invalid
        TDS_VARGEN_CODE_TOO_MANY                        //      Too many variants were generated
};


//      Some of the chars below show up as bars, but they are really different
//      ascii values

//      #define CONSONANTS                                                                      "bdfgh
  => jklmnpqrstvwxzÇâêêÅÄ£¥"¥¡••¶"
#define ANGLO_CONSONANTS                                                                        "bdfgh
  => jklmnprstvwzêêí";•¶"

#define VOWELS                                                                                  "aeiiü
  => o/i|u||"
//      #define IS_CONSONANT(ch)                                                (strchr(CONSONANTS, ch
  => ) != NULL)
#define IS_VOWEL(ch)                                                            (strchr(VOWELS, ch) !=
  =>  NULL)


#pragma warning(disable: 4786)

#include        <use_ansi.h>
#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>
#include        <cstring>
#include  "approx.h"
//      #include <stl.h>

using namespace std;


class   CSimpCodeDistTable;
```

```
// This file contains the class definitions for the Rule Set
// implementation.


// The basic rule class is nothing more than three CNfa objects
// representing the context portions of a rule, and a string for
// the output portion.  (Plus methods to facilitate adding the
// object to arrays)
//
class Rule
{
private:
        void    assign(const Rule& r)   {
                left.Set(r.left.m_Rex,TRUE);
                match.Set(r.match.m_Rex);
                right.Set(r.right.m_Rex);
                output = r.output;
                m_Bad = r.m_Bad;
                doesOutputContainContext = r.doesOutputContainContext;
        }

        bool    doesOutputContainContext;

public:
        CNfa left;
        CNfa match;
        CNfa right;
        CString output;

        BOOL m_Bad;


        Rule() {m_Bad = TRUE; doesOutputContainContext = false;}

        Rule(const Rule& r)     {assign(r);}

        Rule(LPCTSTR l,LPCTSTR m,LPCTSTR r,LPCTSTR o) {Set(l,m,r,o);}

        ~Rule();

        bool    getDoesOutputContainContext() {return doesOutputContainContext;}
        const Rule& operator=(const Rule &r)    {assign(r); return r;}
        int     operator<(const Rule &r)        const {return 1;}
        int     operator==(const Rule &r)       const {return 1;}

        BOOL Set(LPCTSTR l,LPCTSTR m,LPCTSTR r,LPCTSTR o)       {
                output = o;
                //      set if this rule's output field contains a context variable
                //      (e.g. $1, which means the first character of the match context)
                doesOutputContainContext = (strchr((LPCTSTR)output, '$') != NULL);
                m_Bad = !left.Set(l,TRUE) || !match.Set(m) || !right.Set(r);
                return !m_Bad;
        }
};

// class RuleSetInfo is a housekeeping class which keeps track of
// all the rule sets installed into the system.  An array of these
// objects is filled from the INI file (later the registry).  The array
// is kept in the CWbApp object.
//
class RuleSetInfo
{
public:
        BOOL m_bCompiled;
        CString m_SetName;   // what goes in the listbox
        CString m_RulesFile;  // the file of rules for this ruleset
```

```
};

// RuleSet is the actual implementation of the rule set.  Its basicly
// an array of CRule objects (and symbols) and methods for reading them from a file
// and applying them to names.
//

typedef map<CString, CString>   symbols_map_t;
typedef set<CString> string_set_t;
typedef vector<unsigned char>simp_code_vec_t;


//unsigned char digraph_score(const char *string1, const char *string2);
float edit_distance_score(const unsigned char *string1,
                          const unsigned char *string2,
                          CApprox *RecApproxPtr);

class RuleSet
{
private:
        int m_line;
        vector<Rule> m_Rules;

        //      map of the single character expressions, and their codes.
        map<CString, unsigned char>m_secMap;

        //      vector to hold the vectors of (simplified) codes for each rule.
        //      This is parallell to the m_Rules vector.
        vector<simp_code_vec_t> m_simpCodeVec;

        //      vector to hold the simplified Rep Strings for each rule, in the
        //      same order as the m_Rules vector.  We can use the to build (non-
        //      encoded) simplified regular expressions from a name.
        //      This is parallell to the m_Rules vector.
        vector<CString> m_simplifiedRepStrings;

        //      vector to hold the code for each rule.  This is
        //      parallell to the m_Rules vector.  We could have modified
        //      the rules class to include a rec member.
        //      This vector is as big as the number of rules in the ruleset
        vector<unsigned char>m_regExCodeVec;

        //      vector to hold the distinct (and truly different) regex strings
        //      for each code.  The array is indexed by the code, so a lookup
        //      of the string associated with REC 5 is m_replacementRegExStringPtrVec[5].
        vector<CString>m_replacementRegExStringPtrVec;

        int             m_numDistinctCodes;                     //      how many codes were assigned.
        int             m_numDistinctStrings;           //      number of distinct output strings - note that

»                       //      some of these strings may be equivalent.

        symbols_map_t m_symbols;

        BOOL m_bCompiled;
        BOOL m_bLogRules;

        CString m_RulesFile;            // some workspace files just kept as class

        int             FindRule(LPCTSTR word, int wordIndex, CString *pRetString,
                                                                vector<int> *matchingRuleIndex

» Vector);

        void    AddSymbol(LPCTSTR name, LPCTSTR value);
        BOOL    AddSymbol(LPCTSTR symbolLine);
        BOOL    CheckSubSymbol(CString , CString& );
```

```
        BOOL    AddRule(CString strList);
        BOOL    CheckSymbol(CString, CString& );


        //unsigned char         CalcRecCodeCompScore(int code2, vector<CString> *variantsForCode1);
        float CalcRecCodeCompScore(int code2, vector<CString>   *variantsForCode1, CApprox *RecApproxPtr);
        BOOL  CreateCodeComparisonArray();

public:
        void RemoveAll();
        RuleSet(LPCTSTR file=NULL)
        {
                m_bCompiled = FALSE;
                m_RulesFile = file;
                m_bLogRules = FALSE;
        }

        //      array to hold scores that say how closely two RECs are related
        //unsigned char recCodeComparisonArray[255][255];

   // changing this to float to contain the float values that
   // we will now generate, so instead of this array containing
   // values x such that 0 <= x <= 100 it will contain values
   // x such that 0.0 <= x <= 1.0
   float recCodeComparisonArray[256][256];

   void WriteRecMatrixToFile();
   void ReadRecMatrixFromFile(ifstream *);

        ~RuleSet();

        set<CString>    *       getDistinctReplacementStrings();

        BOOL NameMatch(LPCTSTR name1, LPCTSTR name2);

        BOOL Reader(LPCTSTR infile = NULL);
        BOOL Compile(LPCTSTR infile, LPCTSTR outfile,
        BOOL bDebug=TRUE, BOOL bTextOnly=FALSE );

        BOOL IsCompiled() {     return m_bCompiled; }

        BOOL EncodeRules(BOOL buildRECCompArray);

        vector<CString> * genVariants(LPCTSTR name, BOOL devowel,

  »                     BOOL translateName, int truncLen,

  »                     tds_vargen_code &rc);

        CString TranslateWord(LPCTSTR word, BOOL bRemoveSpaces = TRUE);

        BOOL            dumpEncodedRules(ofstream &outfile, int reportMode);
        BOOL            dumpRECCompArray(ofstream &outfile);

//      unsigned        char    **      GetRecCompArray() {return &recCodeComparisonArray[0][0];}
        const   char *          GetRegExForCode(int recCode);


        BOOL            addSimplifiedRules(FILE *simplifiedRulesFile, FILE *logFile, FILE *encodedRulesFile);

        int                     GetSimplifiedCodeArrayForString(LPCTSTR word, unsigned char *simCodeArray,

  »                                     int codeArraySize);
        void            GetSimplifiedRegexForString(LPCTSTR word, CString &simpRegEx);

        CSimpCodeDistTable      *createSimpCodeDistTable(CFeatureDistanceTable *aFeatTable);
```

```
};

#endif // _PARSE_H_
```

```
//     Copyright (C) 1998, Language Analysis Systems Inc.
//
/*-----------------------------------------

   ranker.h: interface for the Ranker class.

   Fall 1997:   Robert Drabek - first version.
   Winter 1998: Robert Drabek - addition of voters and slight changes in
                class definitions.


   1. General:

   This file defines the Ranker class, whose goal is to accept names and
   some information related to these names, and then rank them based on
   certain criteria.

   Three classes are defined here:
       Ranker, (uses RName and RParameters)
       RName,
       RParameters.


   2. Interface:

   Ranker(), init(), submit(), m_scoredNames, setMaxNames(), nameCount()

   Similar to many MFC classes, setting up the Ranker class is done in two
   steps: first the constructor is called, which does very little except set
   an initial limit of zero to the number of names which can be stored, then a
   second member function, init(), is called for initialization of internal
   structures.  One advantage to this is that a Ranker object can be
   reinitialized.  See the sample code below.

   Before submitting the first name to the Ranker, a call to the Ranker::init()
   member function is required, passing it a query, an initialized Paramter
   object, a value for the maximum number of names, and a 256-x-256 float
   table of phonetic feature distances.  The query is of type RName, but note
   that its m_phonetic_score, m_exact_match and several other members are not
   used.

   A second query is actually passed to the init() function, but would normally
   be an empty RName object.  If this is not empty, then the two query names
   should be identical in spelling, but their culture, lead-consonants, and
   leading vowel components will reflect separate culture and rule sets.

   The Ranker::submit() member function takes an argument of type RName; that
   name's
       spelling (string),
       culture (unsigned int),
       phonetic_score (float),
       lead-consonants array (byte *),
       leading vowel (char)
   and
       exact-match (bool)
   fields are expected to be initialized.  This function then gives the name
   a weighted score, and inserts it into the lists appropriately.  The string
   containing the name spelling must be upper case.

   Note that a second, optional, name may be passed to submit().  If passed
   this second name, both names will be scored, and only the name with greater
   weighted score will be inserted.  It is expected that these two names have
   the same spelling, and that the other values were produced by different
   pipes for different languages.

   Also note that the names are actually pointers to RName objects.  These
```

```
   pointers are to be "new" dynamically-allocated objects, and the submit
   function deletes any unused names.

   The public Ranker::m_scoredNames member variable of type
   Ranker::ScoredNames is available after any submission of names to a Ranker
   object.


   RName(), score(), getStr(), getWeightedScore()

   The RName constructor requires
       a string representation of the name's spelling (upper case),
       an unsigned int representing the culture pipe from which the name was
         retrieved,
       a phonetic score precalculated by the matcher,
       an array of opening (leading) phonemes terminated with EOS,
       a char indicating the first vowel of the name,
   and
       a boolean determined by the matcher which indicates it has determined
         this name to be an "exact" match based upon its own criteria.

   The RName::score() member function calculates a weighted score using the
   passed-in query-name object and weight object, and sets its internal
   m_weighted_score value; use getWeightedScore() to access that value.
   Other intermediate scores, such as a digraph score, are also calculated
   and are accessible similarly.


   RParameters(), set*(), get*()

   The RParameters constructor takes no arguments, and assigns some default
   weights to all scoring voters; its values can be reset using the set*
   member functions.


   3. Internals:

   The Ranker uses an STL multiset to store the names.

   In addition, an STL set is used to prevent the same name (its spelling) from
   being added twice (m_scoredNames, m_stringNames).

   Errors encountered by Ranker::submit() are ignored.


   The weighted score calculated by RName::score() is a weighted average.  If
   any errors occur (divide by zero, etc.), a value of 0.0 (zero) is set.


   4. Sample code:

       // Initialize the Ranker.
       RParameters some_params;
         • Optional if default parameters are not desired:
         • some_params.setSpellWt(f1);
         • some_params.setPhoneticScoreWt(f2);
         • other parameters...

       Ranker a_ranker;
       a_ranker.init(a_query, some_params, the_limit, to_bias, fd_matrix);

       // Submit a bunch of names to the Ranker.
       loop in some manner getting a_name, the_culture, a_phonetic_score,
           some_lead_cons, a_lead_vowel and whether_exact {
         RName a_name(a_name, the_culture, a_phonetic_score, some_lead_cons,
                      a_lead_vowel, whether_exact);
```

```
      a_ranker.submit(a_name);
    }

    // View the results list in descending order.
    Ranker::ScoredNames::reverse_iterator i;
    for (i = a_ranker.m_scoredNames.rbegin(); i != a_ranker.m_scoredNames.rend(); i++) {
      // Now you access the name as: (*i)->getStr() [or (*i)->getStr().c_str()].
      // and the score as: (*i)->getWeightedScore()
    }


  5. Maintenance

  If more voters are added, the following needs to be done:

   a. add an RParameters member, m_ppp
   b. initialize m_ppp in the RParameters constructor (and maybe adjust other
      weights)
   c. create setPpp and getPpp functions for RParameters
   d. to RName, add m_ppp_score and getPppScore() and change assign()
   e. in RName::score, set m_ppp_score and add ppp's calculations to the
      numerator and denominator
   f. in RName constructor, initialize m_ppp_score to 0.0f

  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

#if !defined(_RANKER_H)
#define _RANKER_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef byte
typedef unsigned char byte;
#endif

#pragma warning(disable: 4786)

#define NOMINMAX
#include <use_ansi.h>
#include <string>
#include <set>

#include "FeatureDistanceTable.h"

#include "tdssearcher_enums.h"

using namespace std;

// Define RANKER_NAME_SIZE before including this file.
#define RANKER_NAME_SIZE 30

#ifndef RANKER_NAME_SIZE
#define RANKER_NAME_SIZE 20
#pragma message(__FILE__ " : Admonition: RANKER_NAME_SIZE undefined!  Has been set to 20.")
#endif

// These are used in the RName class to indicate existence of leading vowel sounds.
#define R_ALL  'A' // All variants of the name have a leading vowel
#define R_SOME 'S' // Some of the variants have a leading vowel
#define R_NONE 'N' // None of the variants have a leading vowel


class RParameters {
private:
```

```
        float m_phonetic_wt;
        float m_culture_wt;
        float m_lead_cons_wt;
        float m_spell1_wt;
        float m_spell2_wt;
        float m_syllable_wt;
        float m_vowel_wt;
        bool  m_bias;
        float m_threshold;

public:
        // these default weights can be changed as desired
        RParameters() : m_phonetic_wt( 0.65f),
                        m_culture_wt(  0.02f),
                        m_lead_cons_wt(0.02f),
                        m_spell1_wt(   0.09f),
                        m_spell2_wt(   0.18f),
                        m_syllable_wt( 0.02f),
                        m_vowel_wt(    0.02f),
                        m_bias(true),
                        m_threshold(0.70f)
        { }

        void        setPhoneticWt(const float wt) { m_phonetic_wt = wt; }
        void        setCultureWt(const float wt)  { m_culture_wt = wt; }
        void        setLeadConsWt(const float wt) { m_lead_cons_wt = wt; }
        void        setSpell1Wt(const float wt)   { m_spell1_wt = wt; }
        void        setSpell2Wt(const float wt)   { m_spell2_wt = wt; }
        void        setSyllableWt(const float wt) { m_syllable_wt = wt; }
        void        setVowelWt(const float wt)    { m_vowel_wt = wt; }
        void        setBias(const bool b)         { m_bias = b; }
        void        setThreshold(const float th)  { m_threshold = th; }

        const float getPhoneticWt() const { return m_phonetic_wt; }
        const float getCultureWt()  const { return m_culture_wt; }
        const float getLeadConsWt() const { return m_lead_cons_wt; }
        const float getSpell1Wt()   const { return m_spell1_wt; }
        const float getSpell2Wt()   const { return m_spell2_wt; }
        const float getSyllableWt() const { return m_syllable_wt; }
        const float getVowelWt()    const { return m_vowel_wt; }
        const bool  getBias()       const { return m_bias; }
        const float getThreshold()  const { return m_threshold; }
};


class RName;

class less_RName
{
private:
        typedef RName *first_argument_type;
        typedef RName *second_argument_type;
        typedef bool  *result_type;
public:
        bool operator()(const RName *rn1, const RName *rn2) const;
};


class RName {
private:
#define R_MAX_PHONEMES  24
#define R_MAX_LEAD_CONS  6

        // these will be set by the outside world
        string          m_name_str;
```

```
	//	The m_classCulture represents the culture the name was classified
	//	as (or, in the case of a query, possibly the culture the
	//	user specified).  Note that at this time we do not have the
	//	m_classCulture for database names, because the classification
	//	of database names are not stored in the names file.  Thus, a
	//	name that is in the ANGLO file might really be classified as
	//	a different culture, and (since all names are stored in the ANGLO
	//	file without an indication of which culture the name is classified
	//	as by PC-NAS), we do not know if that name is really ANGLO or not.
	//	The m_pipeCulture represents the pipe culture for the name.
	//	For database names, this means "the culture associated with the
	//	file that this name was retrieved from."  For query names, it
	//	describes the extra name info's (start consonants, lead vowel)
	//	associated culture.
	//	The culture voter should compare the m_classCulture of the query
	//	name to the m_pipeCulture of the database name.
	e_tds_culture	m_classCulture;
	e_tds_culture	m_pipeCulture;
	float		m_phonetic_score;
	byte		m_lead_cons[R_MAX_LEAD_CONS + 1];
	char		m_lead_vowel;
	bool		m_is_exact;

	// these will be set by the score() member function
	float		m_culture_score;
	float		m_lead_cons_score;
	float		m_spelling1_score;
	float		m_spelling2_score;
	float		m_syllable_score;
	float		m_vowel_score;

	float		m_weighted_score;

	inline void safesncpy(byte *dest, const byte *src, const size_t len)
	{
		strncpy((char *)dest, (const char *)src, len);
		dest[len] = '\0';
	}

protected:
  void assign(const RName &c);

public:
  // explicit default constructor
	RName();

	// custom constructor
	RName(const string &n,
	      const e_tds_culture classCul,
	      const e_tds_culture pipeCul,
	      const float ps,
	      const byte c[], const char v, const bool e);

	// copy constructor
	RName(const RName &c) { assign(c); }

	// assignment operator
	const RName& operator=(const RName &c) { assign(c); return c; }

	bool operator<(const RName &c) const;
	bool operator==(const RName &c) const;

	void score(const RName &, const RParameters &,
	           const int, CFeatureDistanceTable *,
	           const unsigned int[256]);
```

```
	void makeUpper();

	void setPhoneticScore(const float aScore) { m_phonetic_score = aScore; }

	const string        getStr()            const { return m_name_str; }
	const e_tds_culture getClassCulture()   const { return m_classCulture; }
	const e_tds_culture getPipeCulture()    const { return m_pipeCulture; }
	const bool          getIsExact()        const { return m_is_exact; }

	const float         getPhoneticScore()  const { return m_phonetic_score; }
	const float         getCultureScore()   const { return m_culture_score; }
	const float         getLeadConsScore()  const { return m_lead_cons_score; }
	const float         getSpelling1Score() const { return m_spelling1_score; }
	const float         getSpelling2Score() const { return m_spelling2_score; }
	const float         getSyllableScore()  const { return m_syllable_score; }
	const float         getVowelScore()     const { return m_vowel_score; }

	const float         getWeightedScore()  const { return m_weighted_score; }
};


class Ranker
{
private:
	// An alphabetic list to prevent duplicates of names.
	typedef set< string > names_str;
	names_str m_stringNames;

private:
	// feature differences between any two phonemes
	CFeatureDistanceTable *m_fd_matrix;

	// indices into the vowel distances arrays (which is static in ranker.cpp)
	unsigned int m_vowel_index[256];

public:
	// The list of names sorted in ascending order based upon the weighted
	// scores.
	// Clients should treat this as a "read-only" member.
	typedef multiset< RName *, less_RName > ScoredNames;
	ScoredNames m_scoredNames;

private:
	RName       m_query[2]; // for up to two cultures

	RParameters m_parameters;
	int         m_maxNames;
	bool        m_di_bias;
	int         m_query_syllables[2];

public:
	Ranker();
	virtual ~Ranker();

	void	init(const RName &, const RName &, const RParameters &, const int,
		     const CFeatureDistanceTable *);

	// the names' score values will get changed by this
	void	submit(RName *n1, RName *n2 = NULL);

	const int nameCount() const { return m_stringNames.size(); }

	void	setMaxNames(const int n) { m_maxNames = n; }
};

#endif // !defined(_RANKER_H)
```

```
/*-----------------------------------*/
/*-----------------------------------*/
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// simple code header file for CSimpCodeDistTable class

// The Simple Code Feature Distance Table class
// CSimpCodeDistTable is dependent on the
// CFeatureDistanceTable class

#ifndef _SIMPCODEDISTTABLE_H
#define _SIMPCODEDISTTABLE_H

#include "featuredistancetable.h"
#include "parse.h"


#pragma warning(disable: 4786)

#include        <use_ansi.h>
#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>
#include        <cstring>
#include  "approx.h"
//      #include <stl.h>

using namespace std;

typedef map<CString, unsigned char>simp_codes_map;

class CSimpCodeDistTable
{
// data members
private:
  int m_simp_code_dist_table[256][256];
  bool m_status;


// functions
private:
  void init_table();
  void calc_values(CFeatureDistanceTable *,simp_codes_map *);
  int calc_lowest_score(const char *,const char *,CFeatureDistanceTable *);
  int calc_insertion(const char *, CFeatureDistanceTable *);
  int calc_deletion(const char *, CFeatureDistanceTable *);
  void dump();


public:
  CSimpCodeDistTable(CFeatureDistanceTable *,simp_codes_map *);
  ~CSimpCodeDistTable();
  bool get_status() { return m_status; }
  int get_simpcode_distance(unsigned char chx, unsigned char chy)
  {
    return m_simp_code_dist_table[chx][chy];
  }
};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// simpleEditDistance.h: interface for the CeditDistance class.
//
//////////////////////////////////////////////////////////////////////

#ifndef _SIMPLEED_H
#define _SIMPLEED_H

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef EOS
#define EOS '\0'
#endif

#ifndef STRINGMAX
#define STRINGMAX 30
#endif

class CSimpleEditDistance
{
private:
        int             distanceArray[STRINGMAX+1][STRINGMAX+1];
        float   threshold;

public:
        CSimpleEditDistance();

        virtual ~CSimpleEditDistance();

        void setThreshold(float inThreshold);

        float getThreshold();

        int     getDistanceArrayCell(int i, int j)
        {
                if (i > STRINGMAX+1)
                        return(-1);
                if (j > STRINGMAX+1)
                        return(-2);
                return(distanceArray[i][j]);
        }

        float getDistanceScore(const char *, const char *);
};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                     // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
//      file to hold constant definitions for th TDS project.


#ifndef         TDS_CONSTANTS_DEFFED
#define         TDS_CONSTANTS_DEFFED




#define         TDS_MAX_NAME                                            30

//      how long of an SEC string can we handle
#define         TDS_MAX_SEC_STRING_LEN                  30

//      the length of the code associated with the name
#define         TDS_MAX_NAME_CODE                                   6

//      how big could an entry in the names file be?
//      the entry consists of:
//      name id  (4 bytes)
//      name id (external)  (6 bytes plus a NULL)
//      name                (up to 30 bytes plus a NULL)
//      vowel start flag (1 byte)
//      start consonant list (up to 20 bytes plus a NULL)
//      SEC string (30 bytes plus a NULL)
//
//      This give a total of 4 + 7 + 31 + 1 + 21 + 31 = 95.
//      We round up to 100. just to be safe (This value is used to
//      make sure we read enough bytes to get the whole name.
#define         TDS_MAX_NAME_ENTRY_LEN          100


//      how many names to return per query
#define         TDS_DEFAULT_MAX_NAMES_PER_QUERY 145

//      thresholds
#define         TDS_DEFAULT_NAME_EDITDIST_THRESH                        (float)0.69
#define         TDS_DEFAULT_GROUP_EDITDIST_THRESH                       (float)0.59

//      define the number of cultures and a unique id for
//      each one.  The ids must start with 0 (Anglo should
//      always be 0), and there should not be any gaps in the
//      ids.
#define         TDS_NUM_CULTURES                                                    4


//      define a default file name for the simplified rules file
#define         TDS_SIMPLIFIED_RULES_FILE                                           "tds.simp_rul"

//      define file names for the ANGLO culture
#define         TDS_ANGLO_RULES_FILE                                                    "angl.rul"
#define         TDS_ANGLO_3V_RULES_FILE                                             "angl_3v.rul"
#define         TDS_ANGLO_NAME_FILE                                                         "angl.
 => nam"
#define         TDS_ANGLO_GROUP_FILE                                                    "angl.idx"
#define         TDS_ANGLO_GROUP_NAME_OFFSETS_FILE               "angl.vec"

//      define file names for the ARABIC culture
#define         TDS_ARABIC_RULES_FILE                                                   "arab.rul"
#define         TDS_ARABIC_3V_RULES_FILE                                            "arab_3v.rul"
#define         TDS_ARABIC_NAME_FILE                                                    "arab.nam"
#define         TDS_ARABIC_GROUP_FILE                                                   "arab.idx"
#define         TDS_ARABIC_GROUP_NAME_OFFSETS_FILE      "arab.vec"
```

```
//      define file names for the HISPANIC culture
#define         TDS_HISPANIC_RULES_FILE                                             "hisp.rul"
#define         TDS_HISPANIC_3V_RULES_FILE                                      "hisp_3v.rul"
#define         TDS_HISPANIC_NAME_FILE                                              "hisp.nam"
#define         TDS_HISPANIC_GROUP_FILE                                             "hisp.idx"
#define         TDS_HISPANIC_GROUP_NAME_OFFSETS_FILE    "hisp.vec"

//      define file names for the CHINESE culture
#define         TDS_CHINESE_RULES_FILE                                              "chin.rul"
#define         TDS_CHINESE_3V_RULES_FILE                                           "chin_3v.rul"
#define         TDS_CHINESE_NAME_FILE                                                   "chin.
 => nam"
#define         TDS_CHINESE_GROUP_FILE                                              "chin.idx"
#define         TDS_CHINESE_GROUP_NAME_OFFSETS_FILE             "chin.vec"

//      define the file name that has the group assignments for IPA chars
#define         TDS_GROUP_ARRAY_FILE                                                    "group
 => array.dat"


//      define strings to idenify cultures
#define         TDS_CULTURE_STRING_ANGLO                                "Anglo"
#define         TDS_CULTURE_STRING_ARABIC                               "Arabic"
#define         TDS_CULTURE_STRING_CHINESE                          "Chinese"
#define         TDS_CULTURE_STRING_HISPANIC                         "Hispanic"


//      define file names for the classifier (NAS)
#define         NAS_ARABIC_GN_FILE_NAME                                 "agnames.dbf"
#define         NAS_ARABIC_GN_DIGRAPHS_FILE_NAME        "agdi.dbf"
#define         NAS_ARABIC_GN_TRIGRAPHS_FILE_NAME       "agtri.dbf"
#define         NAS_ARABIC_SN_FILE_NAME                                 "asnames.dbf"
#define         NAS_ARABIC_SN_DIGRAPHS_FILE_NAME        "asdi.dbf"
#define         NAS_ARABIC_SN_TRIGRAPHS_FILE_NAME       "astri.dbf"

#define         NAS_HISP_GN_FILE_NAME                                   "hgnames.dbf"
#define         NAS_HISP_GN_DIGRAPHS_FILE_NAME  "hgdi.dbf"
#define         NAS_HISP_GN_TRIGRAPHS_FILE_NAME "hgtri.dbf"
#define         NAS_HISP_SN_FILE_NAME                                   "hsnames.dbf"
#define         NAS_HISP_SN_DIGRAPHS_FILE_NAME  "hsdi.dbf"
#define         NAS_HISP_SN_TRIGRAPHS_FILE_NAME "hstri.dbf"

#define         NAS_CHIN_GN_FILE_NAME                                   "cgnames.dbf"
#define         NAS_CHIN_GN_DIGRAPHS_FILE_NAME  "cgdi.dbf"
#define         NAS_CHIN_GN_TRIGRAPHS_FILE_NAME "cgtri.dbf"
#define         NAS_CHIN_SN_FILE_NAME                                   "csnames.dbf"
#define         NAS_CHIN_SN_DIGRAPHS_FILE_NAME  "csdi.dbf"
#define         NAS_CHIN_SN_TRIGRAPHS_FILE_NAME "cstri.dbf"


//      distance files
//#define               TDS_IPA_FEATURE_DISTANCE_FILE                           "distance.rul"
#define         TDS_IPA_FLOAT_FEATURE_DISTANCE_FILE             "floatdist.rul"


//      define some identifiers to distinguish between the exact match
//      and fuzzy match retreivals.
#define         TDS_FROM_EXACT_SEARCH                       1
#define         TDS_FROM_SIMILAR_SEARCH             2

//      how many consonants can a name possibly start with?
#define TDS_MAX_START_CONSONANTS            5

//      what name edit distance score do we give for exact phonetic matches
#define         TDS_EDIT_DIST_SCORE_FOR_EXACT_PHON_MATCH            1.0
```

```
#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
//      tds_util.h

#ifndef         TDS_UTIL_DEFFED
#define         TDS_UTIL_DEFFED

#define EOS     '\0'
#define TDS_DEFAULT_WHITESPACE  " \t\r\n"


void    tds_strip(char *aString);
int     tds_unsigned_strcmp(const unsigned char *s1, const unsigned char *s2);



#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
/*      class header for the TDSSearcher class.

        This is an encalsulation of the main processing provided by
        the TDS system.  It contains no GIU specific processing.

        At this time, the file still includes the TDS_CONSTANTS file,
        so that things like file names and lengths can be known.

        Since the class is not tied to any GUI, most functions return
        a bool.  If a bool function returns false, use the getErrMsg()
        function to obtain the error message.

        To use the object:

                Construct an object via the constructor
                Use the set() functions to set whatever values are desired
                call the init() method

        Then for each query:

                call the submitQuery() method, followed by either the
                searchForExactMatches() or searchForSimilarMatches() methods
                (or both).

*/

#ifndef         TDSSEARCHER_DEFFED
#define         TDSSEARCHER_DEFFED

#pragma warning(disable: 4786)

#include        "tdssearcher_enums.h"

//      tdslib includes
#include        "parse.h"
#include        "EditDist.h"
#include        "SimpEditDist.h"
#include        "FeatureEditDistance.h"
#include        "FeatureDistanceTable.h"
#include        "SimpCodeDistTable.h"
#include        "distance.h"

//      pcnas includes
#include <nas.h>

#include        "TDS_CONSTANTS.H"
#include        "GroupData.h"
#include        "ranker.h"

#include        <use_ansi.h>
#include        <iostream>
#include        <fstream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>
#include        <string>

using namespace std;

class   ExtRName;
class   NameRecord;
```

```
class   Ranker;
class   RParameters;
//      **********  typedefs ***************


typedef map<int, NameRecord *>  name_record_map_t;


//      **********  end typedefs ***************



class TDSSearcher
{
public:

        //      a structure to hold info about the current query
        typedef struct tds_query_stats_tag
        {
                int     numCulturesSearched;
  ->    //      we need these so that after the query is done, we remember
                e_tds_culture   secondCulture;          //      how many cultures were searched, and what the
  -> second one was
                int numQueryGroups[2];
                int numNamesWithExactMatchOnGroup[2];
                int numNamesThatMatchedExactly[2];
                int numGroupsCheckedWithEditDist[2];
                int numGroupsThatPassedEditDist[2];
                int numCandidateNamesRetrievedFromSimilarSearch[2];
                int numNamesThatPassedSimilarEditDistance[2];

                void    clear() {
                        numCulturesSearched = 0;
                        secondCulture = TDS_CULT_ANGLO;
                        numQueryGroups[0] = 0;
                        numQueryGroups[1] = 0;
                        numNamesWithExactMatchOnGroup[0] = 0;
                        numNamesWithExactMatchOnGroup[1] = 0;
                        numNamesThatMatchedExactly[0] = 0;
                        numNamesThatMatchedExactly[1] = 0;
                        numGroupsCheckedWithEditDist[0] = 0;
                        numGroupsCheckedWithEditDist[1] = 0;
                        numGroupsThatPassedEditDist[0] = 0;
                        numGroupsThatPassedEditDist[1] = 0;
                        numCandidateNamesRetrievedFromSimilarSearch[0] = 0;
                        numCandidateNamesRetrievedFromSimilarSearch[1] = 0;
                        numNamesThatPassedSimilarEditDistance[0] = 0;
                        numNamesThatPassedSimilarEditDistance[1] = 0;
                }
        } tds_query_stats_t;

        //      constructor.  Note that init() must be called after the
        //      constructor.  The logstream is a stream that the object
        //      uses to log activity.  The caller is responsible for creating
        //      and deleting the stream
        TDSSearcher(ofstream    *aLogstream);
        ~TDSSearcher();

        //      return the current status of the object
        bool    getStatus() {return status;}

        //      return the current error message
        char *getErrMsg() {return errMsg;}
```

```
//      return the current error message
char *getCultureString(e_tds_culture aCulture) {return cultureStrings[aCulture];}

//      function to initialize the object (cause it to read in all of its
//      data.  This must be called before calling submitQuery and any other
//      functions to perform operations.
bool    init();

//      function to submit a new query.  The query name must be passed
//      in.  Before calling this function, caller may wish to adjust
//      some of the query parameters by calling the desired methods -
//      (setNameEditDistanceThresh(), setRankerParms(), etc.)
//      The caller should first call submitQuery(), followed by calls to
//      either searchForExactMatches, searchForSimilarMatches, or both.
//      After this function returns, the caller can call the
//      getQueryStats() method to retrieve things like the culture of
//      the query name and the number of query groups.
bool    submitQuery(const char *qName);         //      initialize the searcher for a new query

//      look for the exact matches
bool    searchForExactMatches();

//      look for similar matches.
bool    searchForSimilarMatches();

//      fills up a vector of pointers to ExtRName objects for
//      the names that were exact matches.  The objects themselves are
//      not owned by the caller and should not be freed.  They are valid
//      until another query is issued, or another name is submitted
//      to the ranker.
void    getResultNamesForExactMatch(vector<ExtRName *> &resultVector);

//      fills up a vector of pointers to ExtRName objects for
//      the names that were similar matches.  The objects themselves are
//      not owned by the caller and should not be freed.  They are valid
//      until another query is issued, or another name is submitted
//      to the ranker.
void    getResultNamesForSimilarMatch(vector<ExtRName *> &resultVector);

//      fills up a vector of pointers to ExtRName objects for
//      the names that were either exact or similar matches.
//      The objects themselves are
//      not owned by the caller and should not be freed.  They are valid
//      until another query is issued, or another name is submitted
//      to the ranker.
void    getAllResultsNames(vector<ExtRName *> &resultVector);

//      returns the number of results we are currently holding
int             getNumResultNames();

//      allows the caller to specify a watch name.  This allows detailed
//      debugging of why a name in the database does or does not come back.
//      Watch name messages are placed in the logfile.
//      Pass in "" or NULL to disable watch name processing
void    setWatchName(const char *watchName);

//      allows the user to specify the address of a variable that will
//      be checked at certain points (during lengthy processing loops)
//      to see if the user has decided to cancel the query.  This is
//      usefull in multi-threaded enviroments where the GUI remains active
//      while another thread is doing processing.
void    setStopVariableAddress(bool *aStopVar) {
        if (aStopVar == NULL)
                stopVar = &falseVar;
        else
```

```
                stopVar = aStopVar;
        }

        //      set the threshold to be used when performing edit distances on
        //      names
        bool    setNameEditDistanceThresh(float aThresh);

        //      set the threshold to be used when performing edit distances on
        //      groups (the TDS Indexes).
        bool    setGroupEditDistanceThresh(float aThresh);

        //      set the weights for the query
        void    setRankerParms(RParameters *aRankerParms);

        //      set the flag that determines if more detailed info should be
        //      added to the log file
        void    setLogDebugInfo(bool aLogDebugInfo) {logDebugInfo = aLogDebugInfo;}

        //      sets how many names (max) we should return.
        bool    setMaxNamesToReturnPerQuery(int anInt);

        //      says if we should try to classify the name, or if the user
        //      will specify the culture (in which case, the culture is
        //      specified as aSpecifiedCult);
        void    setCultureInfo(e_tds_cultureMode aCultMode,
                                                                                        e_tds_culture
  => aSpecifiedCult);

        //      sets what type of adjustment should be performed on the
        //      edit distance calculation
        void    setPreRankerBFMode(e_bf_modes aMode) {preRankerBFMode = aMode;}
        void    setPostRankerBFMode(e_bf_modes aMode) {postRankerBFMode = aMode;}

        void    setFatRankerThreshold(float aFatRankerThreshold)        {fatRankerThreshold = aFatRankerThresh
  => old;}
        void    setFatRankerMaxNames(int aMaxNamesValue)        {fatRankerMaxNames = aMaxNamesValue;}

        //      functions to retrieve data from the object after actions have
        //      been performed
        void    getQueryStats(tds_query_stats_t *aQueryStatsObject) {*aQueryStatsObject = queryStats;}

protected:

        //      storage for the last error message
        char    errMsg[1000 + 1];

        bool    status;
        bool    *stopVar;               //      points to user specified variable that says when
                                                                        //      user wants to
  => stop.
        bool    falseVar;               //      variable that is set to false, and is used
                                                                //      as a default for stopV
  => ar

        //      the the ranker, and a set of default weights
        Ranker                  ranker;
        Ranker                  fatRanker;
        RParameters     defaultRankerParms;

        //      variable to keep track of weather or not the names in the fat
        //      ranker have been retrieved, had their edit distance recalculated,
        //      and been submitted to the final ranker.
        bool                            fatRankerProcessed;

        //      variables that are set by the caller and passed to the ranker
        int                             maxNamesToReturnPerQuery;
```

```
    RParameters     *rankerParms;

    //      other variable that can be set by the user
    e_tds_cultureMode             cultureMode;                  //      should we classify, or will th
=> e user
    e_tds_culture                          specifiedCulture;                    //      if they specif
=> ied a culture, which is it?
    e_bf_modes                                        preRankerBFMode;
    e_bf_modes                                        postRankerBFMode;

    //      fat ranker info
    float   fatRankerThreshold;
    int             fatRankerMaxNames;

    //      log file info
    ofstream        *logstream;
    bool                    logDebugInfo;
    //      ofstream        logstream;

//      variables to support a watch name facility
    char    watchName[TDS_MAX_NAME + 1];
    bool    shouldMatchGroupExact;
    bool    didMatchGroupExact;
    bool    shouldMatchNameExact;
    bool    didMatchNameExact;
    bool    shouldPassGroupEditDist;
    bool    didPassGroupEditDist;
    bool    shouldPassNameEditDist;
    bool    didPassNameEditDist;

    //      variable to hold stats about the current query
    tds_query_stats_t               queryStats;

    //      file names, placed in arrays
    char                    *rulesFileNames[TDS_NUM_CULTURES];
    char                    *rules3VFileNames[TDS_NUM_CULTURES];
    char                    *nameFileNames[TDS_NUM_CULTURES];
    char                    *groupFileNames[TDS_NUM_CULTURES];
    char                    *groupNameOffsetsFileNames[TDS_NUM_CULTURES];

    //      array of culture strings
    char                    *cultureStrings[TDS_NUM_CULTURES];

    //      data objects or files
    RuleSet                 *ruleSets[TDS_NUM_CULTURES];                    //      rule sets
    RuleSet                 *ruleSets3V[TDS_NUM_CULTURES];                  //      3 vowel rule s
=> ets
    FILE                            *nameFiles[TDS_NUM_CULTURES];                   //      name f
=> iles
    GroupData               *groupData[TDS_NUM_CULTURES];                   //      group files

    //      The simplified Rules file.  There is only one of these, and it
    //      is shared by all the rules files.
    FILE                            *simplifiedRulesFile;

char                        m_group_array[256];                 //      group encoding array

    //      retreived data we keep track of throughout the query
    set<unsigned int>exactMatchNameOffsets;             //      offsets of exact matches

    //      map of names that we should send to the Ranker
    name_record_map_t               namesMap;


    //      info about the query
    char                                                queryName[TDS_MAX_NAME + 1];
```

```
    char                                                        spacedQueryName[TDS_MAX_NAME + 1];
    CNfa                                                        queryNfa[2];    //      for possibly 2
=> cultures
    unsigned char                       queryStartIPAChars[2][100 + 1];
    char                                                        queryStartVowelFlag[2];
    set<CString>                        queryGroups[2]; //      for possibly 2 cultures
    vector<CString>             *querySingVowelVariants[2];     //      for possibly 2 cultures
    vector<CString>             *query3VowelVariants[2];        //      for possibly 2 cultures

    unsigned char                       querySECStrings[2][TDS_MAX_SEC_STRING_LEN + 1];
    int                                                             numCulturesToSearch;
=>                  //      how many cultures should
=>                                                                                      //      current
=> query consider
    e_tds_culture                       queryCultures[2];               //      what is index of cultu
=> re we should search?
=>                                                                              //      (within the arr
=> ays we have built)
    bool                                                        threeVowelVariantsTrunced[2];
    bool                                                        singVowelVariantsTrunced[2];

    //      stuff for edit distance
    CEditDistance                                       *editDistancizer;
    CSimpleEditDistance             *simpleEditDistancizer;
    FeatureEditDistance             *featureEditDistancizer;
    CFeatureDistanceTable   *featureDistanceTable;
    CSimpCodeDistTable              *simpCodeDistTable;

    //      thresholds for edit distances
    float   nameEditDistThresh;
    float   groupEditDistThresh;

    //      stuff for pc-nas
    nas             *nameClassifier;



    //      functions

    //      loading data
    bool            loadNameData();
    bool            loadRules();
    bool            load3VRules();
    bool            loadGroups();

    bool            validateQueryName(const char *qName);

    bool            getNameAndEvaluate(int offsetInNameFile, e_tds_culture cultureIndex,

=>          int queryCultureIndex,

=>          int evaluationMethod);
    bool            getNameDataFromFile(unsigned char *nameEntryBuf, int offsetInNameFile,

=>                  e_tds_culture cultureIndex);
    bool            checkExactPhonMatch(char *dbName, e_tds_culture cultureIndex, int queryCultureIndex);
    float           calcBruteForceEditDist(char *dbName);

    bool            initGroupArray(char *groupArrayFileName, char *aGroupArray);
    void            convertVar2Grp(LPCSTR variant, CString *group, bool skipRepeatChars);
    void            computeQueryGroup(int queryCultureIndex, e_tds_culture cultureIndex);

    bool            genQueryVariantsForSingVowel(int queryCultureIndex, e_tds_culture culture, char *query
=> RegEx);
```

```
        bool            genQueryVariantsFor1Vowel(int queryCultureIndex, e_tds_culture culture, char *queryReg
  »  Ex);

//      void            calcShouldMatchOnUCV(vector<CString> &queryVariants);
//      void            calcShouldPassEditDist();

//      void            outputMatchResults();


        void    calcQueryIPAInfo(int queryCultureIndex, e_tds_culture cultureIndex);

        void    emptyNamesMap();

        float   calcSingVowelBruteForceEditDist(char *dbName, int queryCulture,

  »                                                     e_tds_culture culture, float editDistScore);
        float   calc1VowelBruteForceEditDist(char *dbName, int queryCulture,

  »                                         e_tds_culture culture, float editDistScore);

        void    processFatRankerResults();

};


#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//

//      enumerations for the TDSSearcher class

#ifndef TDSSEARCHER_ENUMS_DEFFED
#define TDSSEARCHER_ENUMS_DEFFED

enum    e_tds_culture                   {TDS_CULT_ANGLO, TDS_CULT_ARABIC,                                       TDS_CU
   ↪ LT_CHINESE, TDS_CULT_HISPANIC};
enum    e_tds_cultureMode       {TDS_CULT_MODE_SPECIFY, TDS_CULT_MODE_AUTO};

enum    e_bf_modes      {TDS_BF_MODE_NONE, TDS_BF_MODE_SINGLE,
                                                                TDS_BF_MODE_THREE};


#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
//      master include file for using the TDS system.

#include        "TDSSearcher.h"
#include        "Ranker.h"
#include        "ExtRName.h"
#include        "NameRecord.h"

#include        <tds_util.h>
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _UCVBITMAP_H_
#define _UCVBITMAP_H_


#include <use_ansi.h>
#include <fstream>

#pragma warning(disable: 4786)

#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>

using namespace std;

#include "bitmapizeUCV.h"


typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal1;
  unsigned int bitMapVal2;
} bitmap_entry_t;

/*
typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal;
} bitmap_entry_t;
*/

typedef bitmap_entry_t  *bitmap_entry_ptr_t;


// This file contains the definitions for the UCVBitMap Class
class UCVBitMap
{
public:
        UCVBitMap(const char *fileName, BOOL useMemory);
        ~UCVBitMap();

        //      vector<unsigned int> *  search(vector<unsigned int> *variantBitVals, FILE *ucvIndexFile);
        set<unsigned int> *     search(vector<bits_64_t> *variantBitVals,

  =>            FILE *ucvIndexFile, BOOL *cancelFlag);

        BOOL    getStatus()     {return status;}

        void    setThreshold(float aThreshold)  {threshold = aThreshold;}

        int             getNumBitsForValues(unsigned char *bitString)
        {
                return bitTable[bitString[0]] + bitTable[bitString[1]] +
                                                                        bitTable[bitString[2]]
  =>  +bitTable[bitString[3]] +
                                                                        bitTable[bitString[4]]
  =>  + bitTable[bitString[5]] +
                                                                        bitTable[bitString[6]]
  =>  +bitTable[bitString[7]];
        }

        BOOL    bitCompare(bits_64_t bitVal1, unsigned int *bitVal2, int bitsThatCouldHaveMatched);

private:
        bitmap_entry_t  *mapData;
        FILE                                    *ucvBitMapFile;
        unsigned int            numMapEntries;
        BOOL                                    status;
        BOOL                                    usingMemory;
        unsigned char           bitTable[256];
        float                                   threshold;

        //      BOOL    bitCompare(unsigned int bitVal1, unsigned int bitVal2, int bitsThatCouldHaveMatched);
        void    buildBitTable();

};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _UCVBITMAP_H_
#define _UCVBITMAP_H_


#include <use_ansi.h>
#include <fstream>

#pragma warning(disable: 4786)

#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>.
#include        <set>
#include        <map>
#include        <algorithm>

using namespace std;

#include "bitmapizeUCV.h"


typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal1;
  unsigned int bitMapVal2;
} bitmap_entry_t;

/*
typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal;
} bitmap_entry_t;
*/

typedef bitmap_entry_t  *bitmap_entry_ptr_t;


// This file contains the definitions for the UCVBitMap Class
class UCVBitMap
{
public:
        UCVBitMap(const char *fileName, BOOL useMemory);
        ~UCVBitMap();

        //      vector<unsigned int> *  search(vector<unsigned int> *variantBitVals, FILE *ucvIndexFile);
        set<unsigned int> *     search(vector<bits_64_t> *variantBitVals,
  =>            FILE *ucvIndexFile, BOOL *cancelFlag);

        BOOL    getStatus()     {return status;}

        void    setThreshold(float aThreshold)  {threshold = aThreshold;}

        int             getNumBitsForValues(unsigned char *bitString)
        {
                return bitTable[bitString[0]] + bitTable[bitString[1]] +
                                                                        bitTable[bitString[2]]
  =>  +bitTable[bitString[3]] +
                                                                        bitTable[bitString[4]]
  =>  + bitTable[bitString[5]] +
                                                                        bitTable[bitString[6]]
  =>  +bitTable[bitString[7]];
        }

        BOOL    bitCompare(bits_64_t bitVal1, unsigned int *bitVal2, int bitsThatCouldHaveMatched);

private:
        bitmap_entry_t  *mapData;
        FILE                                            *ucvBitMapFile;
        unsigned int            numMapEntries;
        BOOL                                            status;
        BOOL                                            usingMemory;
        unsigned char           bitTable[256];
        float                                           threshold;

        //      BOOL    bitCompare(unsigned int bitVal1, unsigned int bitVal2, int bitsThatCouldHaveMatched);
        void    buildBitTable();

};

#endif
```

```
//     Copyright (C) 1998, Language Analysis Systems Inc.
//
#ifndef _UCVBITMAP_H_
#define _UCVBITMAP_H_


#include <use_ansi.h>
#include <fstream>

#pragma warning(disable: 4786)

#include        <iostream>
#include        <iterator>
#include        <vector>
#include        <list>
#include        <set>
#include        <map>
#include        <algorithm>

using namespace std;

#include "bitmapizeUCV.h"


typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal1;
  unsigned int bitMapVal2;
} bitmap_entry_t;

/*
typedef struct bitmap_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  unsigned int bitMapVal;
} bitmap_entry_t;
*/

typedef bitmap_entry_t  *bitmap_entry_ptr_t;


// This file contains the definitions for the UCVBitMap Class
class UCVBitMap
{
public:
        UCVBitMap(const char *fileName, BOOL useMemory);
        ~UCVBitMap();

        //      vector<unsigned int> *  search(vector<unsigned int> *variantBitVals, FILE *ucvIndexFile);
        set<unsigned int> *     search(vector<bits_64_t> *variantBitVals,

  =>            FILE *ucvIndexFile, BOOL *cancelFlag);

        BOOL    getStatus()     {return status;}

        void    setThreshold(float aThreshold)  {threshold = aThreshold;}

        int             getNumBitsForValues(unsigned char *bitString)
        {
                return bitTable[bitString[0]] + bitTable[bitString[1]] +
                                                                        bitTable[bitString[2]]
  =>  +bitTable[bitString[3]] +
                                                                        bitTable[bitString[4]]
  =>  + bitTable[bitString[5]] +
                                                                        bitTable[bitString[6]]
  =>  +bitTable[bitString[7]];
        }

        BOOL    bitCompare(bits_64_t bitVal1, unsigned int *bitVal2, int bitsThatCouldHaveMatched);

private:
        bitmap_entry_t  *mapData;
        FILE                                    *ucvBitMapFile;
        unsigned int            numMapEntries;
        BOOL                                    status;
        BOOL                                    usingMemory;
        unsigned char           bitTable[256];
        float                                   threshold;

        //      BOOL    bitCompare(unsigned int bitVal1, unsigned int bitVal2, int bitsThatCouldHaveMatched);
        void    buildBitTable();

};

#endif
```

```
//      Copyright (C) 1998, Language Analysis Systems Inc.

#ifndef _UCVMAP_H_
#define _UCVMAP_H_

//change this manualy for now
//#define MAX_UCV_LEN 19
//      #define MAX_UCV_LEN     3

/*
typedef struct  map_entry_tag
{
        unsigned int            indexOffset;
        unsigned int            numIndexEntries;
        char                                            ucvKey[MAX_UCV_LEN + 1];
}       map_entry_t.
*/

//      the structure below is really not fully descriptive
//      of the real data structure.  The real data structure uses
//      a char[n] member for the ucvKey field.  However, we are
//      required to support files with varying length UCV keys (although,
//      the length is fixed within any given file).  We use the ucvKey
//      field as a placeholder, assigning it a char data type.  When
//      we actually use the data, we reference the address of the field,
//      making it a char *.  Dont forget that the field is NULL terminated.
//      NOTE: this means you can not use the familiar [] method of accessing
//      an individual item in the array
typedef struct map_entry_tag
{
  unsigned int indexOffset;
  unsigned int numIndexEntries;
  char ucvKey;
} map_entry_t;

typedef map_entry_t     *map_entry_ptr_t;


// This file contains the definitions for the UCVMap Class
class UCVMap
{
public:
        UCVMap(const char *fileName, BOOL useMemory, int ucv_length);
        ~UCVMap();

        BOOL    getMapInfoForUCV(const char *ucv, int *indexOffset, int *numIndexEntries).
        BOOL    getStatus()     {return status;}

private:
        map_entry_t                     *mapData;
        FILE                                            *ucvMapFile;
        unsigned int            numMapEntries;
        BOOL                                            status;
        BOOL                                            usingMemory;
        int                                                     ucvLen;
        int                                                     mapItemSize;
        int                                                     ucvKeyOffsetInStructure;

        map_entry_t     *getMapEntryForUCV(const char   *ucv);
        BOOL                    getMapEntryForUCV(const char    *ucv, map_entry_t       *returnMapEntr
  ** y);
};

#endif
```

Name Search Suite of Tool(s)

Functional Requirements/Design Version 1.0

Revised DRAFT (SNAPIGUI ALPHA VERSION 4)

March 19, 1998

1. General Description

LAS is developing a suite of Name Search tools (i.e., APIs) that can be integrated within an existing customer application or can be used to provide the "guts" of a new customer application. The ***LAS Name Search Suite of Tools*** shall : .

- be composed of one or more C++ APIs
- be compatible with any modern platform with a C++ compiler
- provide mechanisms to :
  - compare a query name with one or more candidate names to produce an ordered list of candidate names with the highest probability of representing the same "named" person. This functionality is referred to as the ***Name Comparison Tool(s)*** in the remainder of this document.
  - generate and store intelligent search data for use in extracting relevant subsets of data from large data bases for further evaluation. These mechanisms will facilitate more efficient name searching while ensuring complete and accurate results. This functionality is referred to as the ***Name Extraction Tool(s)*** in the remainder of this document.

The initial offering of the APIs will provide developers with the capability to:

- compare two names to determine the probability that they both represent the same named individual; or
- compare a single query name with a set of candidate names to determine which candidate names are most likely to represent the same named individual.

When a set of candidate names is evaluated, the APIs enable the developer to define the criteria for producing his/her own Results Set. The available options for defining a Result Set include the following:

- an unordered list of all candidate names whose name score exceeds a pre-defined name threshold (e.g., if the threshold = 0, all candidate names will be returned in an unordered list);
- an ordered list of all candidate names whose name score exceeds a pre-defined name threshold (e.g., if the threshold = 0, all candidate names will be returned in an ordered list); or
- an ordered list of the top X candidate names whose name score exceeds a pre-defined name threshold, where X is a number.

1.1 LAS Name Comparison tools

The LAS Name Comparison tools include:

- **NameCheck** - This tool employs multiple evaluation techniques to evaluate and score two names. The NameCheck tool incorporates information regarding variations in spelling, discrepancy in the number of name segments (amount of information included), exclusion of expected information, and positional information in order to establish a name score, which indicates the probability that the two names represent the same individual. The NameCheck tool is controlled by a set of configurable parameters. The NameCheck tool also manages and produces an ordered or unordered list of candidate names with the highest probability of representing the same "named" person, based on the developer-defined criteria for establishing a set of results.
- Various culture-specific tools are available as extensions to the NameCheck tool to perform such functions as the cultural classification of name data (**NameClassifier**), leveling of variations in name data to a single representation (**NameRegularizer**), and the representation of name data based on phonetic similarity (**PhoneticNameKey**).

Version 1.0 of the LAS Name Comparison Tool(s) will establish a baseline of the minimum functionality necessary to perform fuzzy matching on name data. There are two additional enhanced versions of the tool expected to be implemented in-house, prior to producing version 1.0 of a commercially available product. This document defines the functionality to be incorporated into Version 1.0 of the tool, and in some cases, describes why certain decisions were made regarding specific functionality. The document also notes areas for planned future enhancement.

1.2 LAS Name Extraction tools

The LAS Name Extraction tools include:

- **An Intelligent Search Data Generator (ISDG)** which generates one or more search data values that facilitate extraction of relevant information from a data base for further comparative analysis. This tool is a critical component of any search system that must search large volumes of data to locate similar name data. It is not feasible to retrieve and evaluate every name record in a data base to determine its relevance to a query name. The ISDG provides a motivated method for retrieving all relevant information from a data base while reducing the amount of non-relevant information retrieved. This tool can provide significant performance improvements while also ensuring an accurate and complete name search.
- Various culture-specific tools are available as extensions to the ISDG to perform such functions as the cultural classification of name data (**NameClassifier**), leveling of variations in name data to a single representation (**NameRegularizer**), and the representation of name data based on phonetic similarity (**PhoneticNameKey**).

Note that in the current version of this document, there is no further discussion of the Name Extraction Tool(s). These tools will be developed in the future.

2. Perform Error Handling

2.1 Functionality

The tool shall establish a standard list of error codes and their associated text descriptions.

Each function call shall return an error code whenever error checking is appropriate.

The tool shall also provide the capability for the developer to retrieve the text associated with the error code.

The following is a list of the error codes and their meaning:

Error Code	Meaning

3. Produce Linguistic Trace

3.1 Functionality

Version 1.0 will not provide any Linguistic Trace functionality.

4. Accept Input Name Data

4.1 Input Parameters

4.1.1 Functionality

The tool shall verify that all input parameters have valid values as defined in the table below.

The tool shall support several query types (i.e., pre-defined sets of parameters) to facilitate searching the data based on different cultural or other linguistic perspectives.

Certain combinations of these parameters provide better results when addressing known combinations of cultural and/or other linguistic issues.

The tool shall provide the developer with the capability of selecting (defining) one of the API-defined query types.

The tool shall also provide the developer with the capability of modifying any or all of the selected query set parameter values.

At a minimum the tool shall support the following query types :

- Generic;
- Anglo;
- Arabic;
- Chinese;
- Hispanic;
- Korean; and
- Russian.

The tool shall ***not*** allow parameters to be changed in the middle of processing a query. (*see design notes below*).

The set of parameters included in a query type shall include:

Name Field	Parameter	Valid Range / Set of Values	Generic Default Value	Anglo Default Value (Eng/US)	Arabic Default Value (Egypt)	Chinese Default Value (China)	Hispanic Default Value (Mexico)	Korean Default Value (Korea)	Russian Default Value (Russia)
SN	SurnameCheckInitial	{T, F}	F	F	T	F	T	F	T
SN	SurnameCheckVariant	{T, F}	T	T	F	T	T	T	F
SN	SurnameCheckBias	{T, F}	F	F	F	F	F	F	T
SN	SurnameCheckUnknownNotExist	{T, F}	F	F	F	F	F	F	F
SN	SurnameCheckCompressed	{T, F}	F	F	T	F	T	F	F
SN	SurnameAnchorSegment	{first, last, none}	none	none	none	none	first	none	none
SN	SurnameCheckTAQ	{off, remove, score}	score	score	score	score	score	score	score
SN	SurnameMode	{highest, average, lowest}	average	average	average	average	average	average	average
SN	SurnameExactInitialMatchScore	{0.0, 0.1, ... 1.0}	1.0	1.0	1.0	0	1.0	0	1.0
SN	SurnameInitialScore	{0.0, 0.1, ... 1.0}	0	0	.85	0	.85	0	.85
SN	SNV-Score *	{0.0, 0.1, ... 1.0}	-	-	-	-	-	-	-
SN	SurnameOutOfPositionFactor	{0.0, 0.1, ... 1.0}	.60	.60	.90	1.0	.60	.63	.80
SN	SurnameAnchorFactor	{0.0, 0.1, ... 1.0}	0	0	0	0	.70	0	0
SN	SurnameTAQDisregardAbsentFactor	{0.0, 0.1, ... 1.0}	.80	.80	.80	.80	.80	.80	.80
SN	SurnameTAQDeleteAbsentFactor	{0.0, 0.1, ... 1.0}	.90	.90	.90	.90	.90	.90	.90
SN	SurnameTAQDeleteFactor	{0.0, 0.1, ... 1.0}	.85	.85	.85	.85	.85	.85	.85
SN	SurnameTAQDisregardFactor	{0.0, 0.1, ... 1.0}	.70	.70	.70	.70	.70	.70	.70
SN	LastNameUnknownScore	{0.0, 0.1, ... 1.0}	.60	.60	.60	.60	.60	.60	.60
SN	NoLastNameScore	{0.0, 0.1, ... 1.0}	.65	.65	.65	.65	.65	.65	.65
SN	SurnameCompressedScore	{0.0, 0.1, ... 1.0}	.90	.90	.90	.90	.90	.90	.90
SN	SurnameThreshold	{0.0, 0.1, ... 1.0}	.50	.50	.63	.70	.60	.63	.62
SN	SurnameWeight	{0.0, 0.1, ... 1.0}	1.0	1.0	.80	1.0	1.0	1.0	1.0

GN	GivenNameCheckInitial	{T, F}	T	T	T	F	T	F	T
GN	GivenNameCheckVariant	{T, F}	T	T	F	T	T	T	F
GN	GivenNameCheckBias	{T, F}	F	F	F	F	F	F	F
GN	GivenNameCheckUnknownNotExist	{T, F}	F	F	F	F	F	F	F
GN	GivenNameCheckCompressed	{T, F}	F	F	T	F	T	F	F
GN	GivenNameAnchorSegment	{first, last, none}	none	none	none	none	none	none	first
GN	GivenNameCheckTAQ	{off, remove, score}	score	score	score	score	score	score	score
GN	GivenNameMode	{highest, average, lowest}	average	average	average	lowest	average	average	highest
GN	GivenNameExactInitialMatchScore	{0.0, 0.1, ...1.0}	1.0	1.0	1.0	0	1.0	0	1.0
GN	GivenNameInitialScore	{0.0, 0.1, ...1.0}	.85	.85	.85	0	.85	0	.85
GN	GNV-Score *	{0.0, 0.1, ...1.0}	-	-	-	-	-	-	-
GN	GivenNameOutOfPositionFactor	{0.0, 0.1, ...1.0}	.60	.60	.70	0	.60	.69	.65
GN	GivenNameAnchorFactor	{0.0, 0.1, ...1.0}	0	0	0	0	0	0	.60
GN	GivenNameTAQDisregardAbsentFactor	{0.0, 0.1, ...1.0}	.80	.80	.80	.80	.80	.80	.80
GN	GivenNameTAQDeleteAbsentFactor	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
GN	GivenNameTAQDeleteFactor	{0.0, 0.1, ...1.0}	.85	.85	.85	.85	.85	.85	.85
GN	GivenNameTAQDisregardFactor	{0.0, 0.1, ...1.0}	.70	.70	.70	.70	.70	.70	.70
GN	FirstNameUnknownScore	{0.0, 0.1, ...1.0}	.60	.60	.60	.60	.60	.60	.60
GN	NoFirstNameScore	{0.0, 0.1, ...1.0}	.65	.65	.65	.65	.65	.65	.65
GN	GivenNameCompressedScore	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
GN	GivenNameThreshold	{0.0, 0.1, ...1.0}	.50	.50	.63	.70	.60	.69	.60
GN	GivenNameWeight	{0.0, 0.1, ...1.0}	.80	.80	1.0	.80	.80	.80	.80
SN+GN	NameThreshold **	{0.0, 0.1, ...1.0}	.60	.60	.63	.70	.60	.66	.61

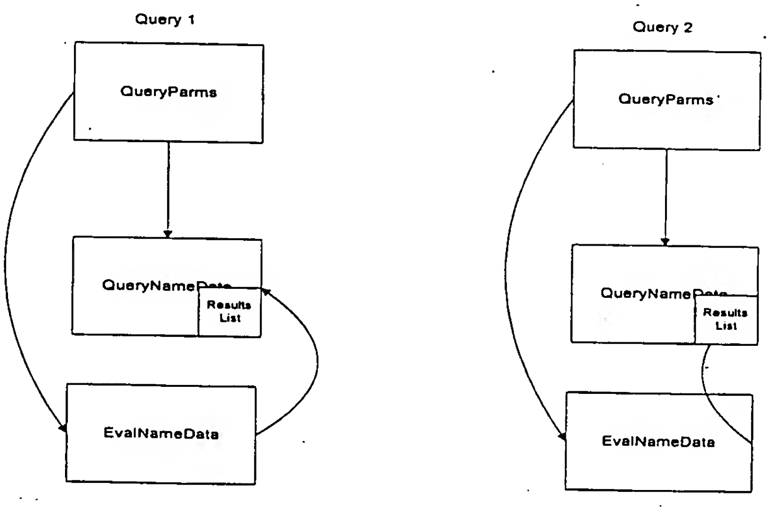
(Note that the values of the parameters that exist in DNC were mapped into their related parameter value in this set of default values. All new parameters were assigned a "best guess" value at this point. Some adjustments were made to the DNC GNOOPS + SNOOPS parameters).

* SNV-Score and GNV-Score values are not included in this table since the scores are actually associated with a specified variant pair, and are contained in the SURNAME-VARIANT and GIVEN-NAME-VARIANT Tables, respectively. The developer can not override the SNV-Score and GNV-Score through the API. Changes to these scores must be made through a separate VariantManager utility. Refer to the sections on SN Variants and GN Variants for more details.

** NameThreshold was calculated by scoring $\frac{\text{(SurnameThreshold*SurnameWeight)+(GivenNameThreshold*GivenNameWeight)}}{\text{(SurnameWeight+GivenNameWeight)}}$

4.1.2 Design Notes

1. Should the PARMS object be shared or copied to each query-name and evaluation-name object? The following diagram illustrates how all PARMS data would have to be copied and carried with every query object if we are to allow PARMS to be changed in the middle of processing without affecting the queries that are already in progress. This could result in significant overhead (i.e., memory processing). If the PARMS object is shared then changes to any PARMS in the middle of processing could potentially affect processing that is already in progress. An example of why we would want to change the PARMS in the middle of processing is : We want to re-compare the same query name using the same candidate names but with a different set of parameters. If we do not allow the PARMS to change, then the developer would need to re-call the tool and have the tool re-process the query name and the candidate names in order to compare the names with different parameter settings. The lists of TAQs and Given Name Variants are not considered update-able in the middle of processing. The PARMS that are considered update-able are those PARMS that the developer sets when the tool is called to perform a comparison.



4.1.3 Future Version Notes

1. The tool shall provide the developer with the capability to modify existing or establish new query sets of pre-defined parameters. (Parameter Definition Application).

2.

4.2 Input Name Model

4.2.1 Functionality

The tool shall provide separate function calls for the following name models :

- Given Name + Middle Name + Surname (GMS)
- Given Name + Surname (GS)
- Name (N)
  - Name
  - Surname, Given Name (SN, GN)

The developer will call the desired function and provide the relevant string values in the appropriate function parameters.

The tool shall accept empty string parameter values to the name model function calls. This functionality will be provided to support a customer data base that allows null values or empty strings in any of the fields (e.g., middle name) defined in their name model.

Because the tool itself utilizes the GS model, the most efficient and accurate results will be provided if the GS model is received as input.

4.2.2 Design Notes

1. We selected a function call approach as opposed to passing in a single name string with delimiters as the function call approach will:
   - be easier for a developer to determine which call is appropriate for the business need;
   - not require the developer to identify or understand irrelevant parameters;
   - not require the developer to incorporate irrelevant parameters into their application code; and
   - be more efficient.

4.2.3 Future Version Notes

1. The tool may support the following additional name models/functions :
   - Given Name + Middle Name + Surname + Maiden Name (GMSM)
   - Given Name + Surname + Maiden Name (GSM)

2. The tool may also utilize other name models besides the GS model, if deemed beneficial.

5. Preprocess Name Data

5.1 Functionality

The tool shall preprocess input name data using the following techniques; in the order listed below:

- Identify and parse input name data into given name and surname (name fields)
    - *in future, may use High Frequency Surname data to define surname field*
    - *in future, may move titles and qualifiers into given name field*
    - *in future, unknown and non-existent name values may be used to define given name and surname fields*
- Validate input name data
- Convert name data to UPPER case
    - *in future, this step may move after the TAQ processing (after conjoined TAQ processing is implemented)*
- Preprocess Segmentation and Removal markers (Noise data)
- Parse name fields into name segments
- Identify and process unknown and non-existent name values (e.g., "FNU", "LNU")
- Identify and process minor name parts (e.g., Titles, Affixes, Qualifiers)
    - *in future, identify gender, if applicable*
    - *in future, may identify and process morphological endings separate from TAQs*
- Identify number of segments in name fields
- Identify and process Given Name Variants (Query Only)
    - *in future, identify gender, if applicable*
- Identify and process Surname Variants (Query Only)

5.1.1 Identify and parse input name data into given name and surname (name fields)

5.1.1.1 Functionality

If name data are received in a name model other than GS, then the name data shall be parsed into a GS model.

If the GMS name model is provided, then the internal given name field shall be constructed by placing the input given name in the same field with the input middle name, and the internal surname field shall be set equal to the input surname field.

If the name model does not distinguish the data beyond a single name field (N model), then the tool shall accept the last (i.e., right-most) name segment in the name field as the surname, and place all other name segments in the given name field (e.g., Name: *Jose Garcia Gomez* → Given Name: *Jose Garcia* Surname: *Gomez*). The tool shall recognize the first comma in the Name field (N model) to represent a SN, GN model. The tool shall move the data to the left of the comma into the SN field, and the data to the right of the comma into the GN field (the comma shall be removed from further processing).

The tool shall retain the original form of the parsed Given Name field and Surname field for subsequent processing (i.e., Determine GivenNameCompressedScore, Determine SurnameCompressedScore, and Provide Results information to return to user).

5.1.1.2 Future Version Notes

1. The tool may move all Titles, and Qualifiers into the Given Name field.
2. The tool may utilize High Frequency (HF) Surname and TAQ data to determine the structure of the name data (e.g., two HF Hispanic Surnames found in a Name string can be used to identify a Hispanic Surname; ABU means "father of" and BIN means "son of" in Arabic names. Reference the example above, Name: "Jose Garcia Gomez" → Given Name: "Jose" Surname: "Garcia Gomez").
3. With Arabic names, the tool may move all name segments other than the first name segment (presumably the first given name) into the SN field – clearly this functionality can not be implemented until the NameClassifier tool is made available. Other criteria may be used such as TAQ values.
4. If HF Surnames are found anywhere in the GN field, the tool may move them to the SN field. This may prove beneficial to handling multi-segment surnames such as those that occur in the Hispanic naming system. Sometimes HF Surnames appear in the GN field because they are aliases, so we must be careful with this.
5. The tool may utilize "NFN", "NMN", and "NLN" values when creating the name fields. The only contents of the SN field should be "NLN" or "LNU" if they occur anywhere in the name. If "NFN", "FNU", "MNU", or "NMN" occur, then they should occur only in the GN field. However, additional values may be allowed in the GN field. If name models other than the GS model are utilized within the tool itself, the tool may support more sophisticated processing of these values (e.g., the GMS)
6. Since we decided that we would not handle Maiden name data at this time, there is no special handling of middle name data at this time. Future versions of the tool may use gender data, if available, to manipulate the middle name when dealing with female data – only when dealing with Anglo names, however.

5.1.2 Validate input name data

5.1.2.1 Functionality

The tool shall assume that all name data are person names.

The tool shall accept name data (given name plus middle name plus surname) up to 255 character length total. Thus, the tool shall support Given Name <= 255 characters, MN <= 255 characters, and

Surname <= 255 characters, since any one of these fields may contain all of the name data. The tool shall truncate any data that exceeds the specified maximum character length.

The tool shall accept any case as input (i.e., upper, lower, or mixed case).

The tool expects roman characters (i.e., alphabetic, numeric, and punctuation markers) as input, however it shall accept both roman and non-roman characters and process them in the same manner.

Through Version 1.0, the tool shall not support double-byte character sets.

5.1.2.2 Future Version Notes

1. The tool may recognize and support double-byte character sets (e.g., unicode).

5.1.3 Convert name data to UPPER case

5.1.3.1 Functionality

The tool shall change all name data to upper case.

5.1.3.2 Design Notes

5.1.3.3 Future Version Notes

1. The tool may reference the TAQ Table to process the name data more intelligently prior to converting the data from mixed case to upper case.. TAQ data may facilitate more intelligent segmentation of the name data (e.g., "VanDerMinten" → "Van Der Minten" or "DAngelo" → "D Angelo"). The tool may look up the values "Van", "Der", and "Minten" in the TAQ Table; if found, then the tool shall identify the values as TAQs and process them appropriately. If not found, the tool shall rejoin any remaining non-TAQs and then convert them to upper case.

2. The tool may also reference a High Frequency Surname Table to process the name data more intelligently prior to converting the data from mixed case to upper case. High Frequency Surname data may facilitate more intelligent segmentation of the name data (e.g., "GarciaGomez" → "Garcia Gomez"). The tool may look up the Surnames "Garcia" and "Gomez" in the High Frequency Surname Table; if found, then the tool shall identify the Surnames as High Frequency Surnames and process them appropriately. If not found, the tool shall rejoin any remaining non-High Frequency Surnames and then convert them to upper case.

3. The tool may utilize case information to process the name data more intelligently in conjunction with TAQ and HF Surname processing (e.g., DeLaÇruz → De La Cruz or GarciaGomez → Garcia Gomez) to assist in determining and parsing name segments.

5.1.4 Preprocess Segmentation and Removal markers (Noise data)

5.1.4.1 Functionality

The tool shall recognize non-alphabetic characters received in the name model.

The tool shall reference a list of replaceable single character non-alphabetic data values to determine what process if any is required for the character encountered.

The tool shall replace each item identified in the list of single character values (i.e., punctuation and/or numbers) with their designated single character REPLACEMENT VALUE as identified in the replacement list.

The tool shall only allow a marker to be defined as either a removal marker or a segmentation marker. If a marker is defined as both a removal marker and a segmentation marker, then the tool shall recognize the marker as a removal marker.

The Table below illustrates the default contents of the replacement list.

Note that the REPLACEMENT-VALUE in the table below is a textual description of an empty string (designated by NIL) or a space (designated by BLANK), which is provided for ease of reading, and is not necessarily representative of the physical representation of the list referenced by the tool.

The tool shall recognize a list of single character markers that indicate the end of a name segment / beginning of a new name segment by replacing the segmentation markers with a space (designated by BLANK in the table below).

The tool shall recognize standard segmentation delimiters such as tab, new line, carriage return, etc. without them being explicitly entered into the segmentation list.

The tool shall recognize a list of markers that are designated for removal by deleting the values entirely from the name field (i.e., mapping each removal value to an empty value or no value; designated by NIL in the table below).

The tool shall provide default lists of removal markers and segmentation markers.

The tool shall allow the developer to provide a custom removal list.

The tool shall also allow the developer to provide a custom segmentation list.

The tool shall accept an empty removal list to indicate turning off removal processing. If a BLANK is included in the removal list, multiple segment name fields shall be recognized by the tool as a single segment.

The tool shall accept an empty segmentation list to indicate turning off segmentation processing. If an empty segmentation list is provided, multiple segment name fields shall be recognized by the tool as a single segment.

If the developer does not provide a segmentation list at all, the tool shall utilize its own default segmentation list.

If the developer does not provide a removal list at all, the tool shall utilize its own default removal list.

VALUE	REPLACEMENT VALUE
BLANK	BLANK (segmentation value)
!	NIL (removal value)
"	NIL (removal value)
#	NIL (removal value)
$	NIL (removal value)
%	NIL (removal value)
(	NIL (removal value)
)	NIL (removal value)
*	NIL (removal value)
+	NIL (removal value)
-	BLANK (segmentation value)
.	NIL (removal value)
/	NIL (removal value)
:	NIL (removal value)
;	NIL (removal value)
<	NIL (removal value)
=	NIL (removal value)
>	NIL (removal value)
?	NIL (removal value)
@	NIL (removal value)
[	NIL (removal value)
\	NIL (removal value)
]	NIL (removal value)
`	NIL (removal value)
{	NIL (removal value)
|	BLANK (segmentation value)
}	NIL (removal value)
'	NIL (removal value)
0	NIL (removal value)
1	NIL (removal value)
2	NIL (removal value)
3	NIL (removal value)
4	NIL (removal value)
5	NIL (removal value)
6	NIL (removal value)
7	NIL (removal value)

8	NIL (removal value)
9	NIL (removal value)

5.1.4.2 Design Notes

1. We decided not to implement a string translation Table or regular expressions at this time for three primary reasons:

   - We determined that it would be easier to implement single character replacements at this time.
   - We were not able to determine that there really is a need at this point to handle more than single character replacements, as rewrite rules and other techniques such as the TAQ processing (separate-if-conjoined, disregard, and delete) satisfied all of the examples we could come up with for string replacements. The example of differences in handling apostrophes was resolved by TAQ processing.
   - Although regular expressions may provide the most flexible solution, our current regular expression routines are restricted to the windows environment and we will need more time to investigate alternative approaches prior to their implementation in the tool.

2. Even so, in the future, the tool may pre-process punctuation through a string replacement Table or through regular expressions to enable us to replace contextualized punctuation with some string value, if necessary. These preprocessing rules will probably be culture-specific, and therefore, will also require that the tool support culture-specific processing.

3. Special non-roman characters that often appear intermixed with roman characters, (e.g.., □, □, □, □), will probably be handled with rewrite rules via regular expressions by other functions yet to be defined for the tool.

5.1.4.3 Future Version Notes

1. We may want to look further at " " and ( ) because these values are sometimes used to designate an alias or nickname when they appear in either the SN field or the GN field (e.g., PITRA **"PETROFF"**, SANTANA ANDRE or EMAN, JAN **(HENNY)** H.).

5.1.5 Parse name fields into name segments

5.1.5.1 Functionality

Name fields shall be parsed into name segments with remaining punctuation in tact (i.e., any punctuation not processed in 2.4 shall be left in tact in the name data).

The tool shall define a name segment as a string of text surrounded by white space.

The tool shall support up to 10 segments in both the SN and GN fields prior to removal of TAQs.

If more than 10 segments are provided in Version 1.0, any additional segments will simply be excluded from the evaluation process.

The tool shall support name segments up to 30 characters in length.

5.1.5.2 Future Version Notes

1. The tool may segment two HF names if found conjoined.

5.1.6 Identify and process unknown and non-existent name values

5.1.6.1 Functionality

The tool shall recognize the following special characters which indicate unknown or non-existent name segment values:

- "FNU" - representing first name unknown;
- "MNU" - representing middle name unknown;
- "LNU" - representing last name unknown;
- "NFN" - representing no first name;
- "NMN" - representing no middle name; and
- "NLN" - representing no last name.

The tool shall replace any GN segment containing "FNU", "MNU", "NFN", and "NMN" with an empty GN segment and tag the segment as "unknown" or "not-exist".

The tool shall replace any SN segment containing "LNU" and "NLN" with an empty SN segment and tag the segment as "unknown" or "not-exist".

The tool shall tag all other SN or GN segments as "known".

5.1.6.2 Design Notes

1. Tagging these special values will enable the tool to use this information during the evaluation process.

2. The tool assumes that the name field processing has already addressed the issue that "LNU" and "NLN" should not appear in the GN field and that "FNU", "MNU", "NFN", and "NMN" should not appear in the SN field. Thus, there is no special handling done at this point in the process.

5.1.7 Identify and process minor name parts (e.g., Titles, Affixes, Qualifiers)

5.1.7.1 Functionality

The tool shall identify Titles, Affixes (prefixes, suffixes, infixes), and Qualifiers (TAQs) in the name data, by referencing a Table of single segment TAQs (i.e., no complex TAQs, such as *al din*) that are defined within the context of a specified cultural group or partition. Note that the TAQ Table does not include punctuation in Version 1.0. Future versions may include punctuation such as *O'*.

5.1.7.1.1 TAQ Table

The TAQ Table shall contain Titles, Affixes, and Qualifiers that are described in context of a specified cultural group or partition (i.e., CULT-AFF-ID). In Version 1.0 of the tool, the TAQ table shall include a "Generic" partition (i.e., non-culture specific), as well as Anglo, Arabic, Chinese, Hispanic, Korean, and Russian. The table below lists the cultural partitions that will be included in Version 1.0:

CULT-AFF-ID	CULTURAL-AFFINITY
A	Arabic
C	Chinese
E	Anglo
G	Generic
H	Hispanic
K	Korean
R	Russian

A "Generic" partition shall be composed of the most commonly occurring TAQ values that can be evaluated as a TAQ value regardless which culture is being evaluated. In other words, "Generic" TAQs frequently occur in multiple cultures. For example, "MR" and "PHD" often occur in a variety of multi-cultural names. Titles and Qualifiers readily fall into the "Generic" category. Affixes are less likely to occur in the "Generic" category.

In most cases, TAQ values that are included in the "Generic" partition will not be included in a cultural partition, even though they may be associated with a specific culture. Exceptions will occur when the TAQ definition (TAQ-TYPE-CODE, GENDER, SEPARATE-IF-CONJOINED, SN-PROCESS-ID, or GN-PROCESS-ID) or TAQ processing is distinct in the different cultures. For example, "SR" is an abbreviation of the Hispanic title "Senor" as well as a Generic qualifier indicating "senior or the first".

The TAQ Table shall not be modifiable by the developer or user in Version 1.0.

A separate data base utility will be developed to generate code representing the contents of the TAQ Table which is currently stored in MS Access.

An example of the contents of the TAQ Table is provided below:

CULT-AFF-ID	TAQ	TAQ-TYPE-CODE	SEPARATE-IF-CONJOINED	GENDER	SN-PROCESS-ID	GN-PROCESS-ID
A	AABD	P	0	U	DIS	DIS
A	AAL	P	0	U	DIS	DIS
A	ABA	P	0	M	DIS	DIS
A	ABBD	P	0	U	DIS	DIS
G	ABD	P	0	U	DIS	DIS
A	ABDAL	P	0	U	DIS	DIS
A	ABDAN	P	1	U	DIS	DIS
A	ABDAR	P	1	U	DIS	DIS
A	ABDAS	P	0	U	DIS	DIS
G	ABDEL	P	0	U	DIS	DIS
A	ABDEN	P	1	U	DIS	DIS
A	ABDER	P	1	U	DIS	DIS
A	ABDES	P	1	U	DIS	DIS

Each TAQ shall be classified as a P-Prefix, S-Suffix, T-Title, I-Infix, Q-Qualifier (TAQ-TYPE-CODE). *Note that at the present time, we have no Infixes in the Table.*

Each TAQ shall be classified as to whether or not the TAQ shall be recognized and separated from a stem, if the TAQ occurs conjoined with the stem (SEPARATE-IF-CONJOINED; 1="T", 0="F"). Version 1 of the tool shall not process SEPARATE-IF-CONJOINED.

Each TAQ shall be assigned a gender (GENDER) of either "M" - male, "F" - Female, or "U" - Undefined. Version 1 of the tool shall not process GENDER.

Each TAQ shall be classified distinctly for the SN field (SN-PROCESS-ID) and GN field (GN-PROCESS-ID) as either a DELETE TAQ ("DEL") or a DISREGARD TAQ ("DIS").

5.1.7.1.2 TAQ Processing

If the parameter GivenNameCheckTAQ = "off", the tool shall not perform any TAQ processing in the GN field.

If the parameter SurnameCheckTAQ = "off", the tool shall not perform any TAQ processing in the SN field.

If the parameter GivenNameCheckTAQ = "remove" or "score", the tool shall perform TAQ processing in the GN field as described below.

If the parameter SurnameCheckTAQ = "remove" or "score", the tool shall perform TAQ processing in the SN field as described below.

The tool shall consider TAQs to occur anywhere within the GN field or GN segment, if the GivenNameCheckTAQ = "remove" or "score".

The tool shall consider TAQs to occur anywhere within the SN field or SN segment, if the SurnameCheckTAQ = "remove" or "score".

The tool shall recognize and remove all TAQs from the GN and SN name fields, but retain the actual value of the TAQ as well as the classification of the TAQ as a DELETE TAQ or DISREGARD TAQ. Each TAQ's set of retained information shall be associated with the TAQ's related name segment for use in the evaluation process.

The tool shall recognize TAQs via the following steps:

- First, the tool shall look up each GN or SN segment to determine whether it is included in the TAQ Table for the appropriate cultural perspective (CULT-AFF-ID) as defined by the API-defined query type.
    - If the GN or SN segment is included in the subset of the TAQ Table associated with the selected cultural perspective, the tool shall process the TAQ according to the culture-specific definition, as described below.
    - If the GN or SN segment is not included in the subset of TAQ Table associated with the selected cultural perspective, and the selected cultural perspective is not "Generic", then the tool shall look up each GN or SN segment to determine whether it is included in the "Generic" subset of the TAQ Table.
        - If the GN or SN segment is included in the "Generic" subset of the TAQ Table, the tool shall process the TAQ according to the culture-specific definition, as described below.
        - If the GN or SN segment is not included in the "Generic" subset of the TAQ Table, the tool shall perform no additional TAQ processing for this GN or SN segment.

The tool shall utilize the culture-specific definition of the TAQ (information in the TAQ Table about each TAQ) to determine its related segment in the following manner:

- Stems are defined as any segment whose value is not defined as a TAQ (i.e., is not included in the TAQ Table);
- Any TAQ located to the left of the first stem will be associated with the first stem;
- Any TAQ located to the right of the final stem will be associated with the final stem; and
- For medial TAQs, the following rules shall apply:
    - Find the rightmost suffix (as defined in the TAQ Table) following a stem;

- That suffix and any other TAQ preceding it shall be associated with the preceding stem; and

- Any remaining TAQs shall be associated with the next stem.

The following example illustrates how TAQs will be associated with stem segments.

DOKTOR	ABD	EL	**RAHMAN**	**NOOR**	EL	DIN	ABD	EL	**KADIR**
T1	P1	P2	STEM1	STEM2	P3	S1	P4	P5	STEM3

T1 P1 P2 are all associated with STEM1
P3 S1 are associated with STEM2
P4 P5 are associated with STEM3

If every name segment contained in a name field is identified as a TAQ, then the tool shall associate all of the TAQs with a single empty segment.

TAQs may occur conjoined with a name stem, as is the case with DeLaCruz, O'Connor, and MacDougal, or they can occur as disjoined segments within a name, as in De La Cruz, O' Connor and Mac Dougal. *Version 1.0 of the tool shall not recognize or process conjoined TAQs.*

5.1.7.2 Design Notes

1. A selected subset of the current corporate TAQ Table will be designated for inclusion in the product TAQ Table.

2. The tool will not recognize complex TAQs, such as "al din". Previous prototypes for the State Department (legacy ANA) have supported complex TAQs for reasons that are not relevant to this tool. For more information on this issue, refer to the corporate linguistic data repository TAQ documentation.

3. The tool will accept empty strings in the SN and GN fields, so there is no special processing to handle the situation when only TAQ(s) occur in one of the name fields.

5.1.7.3 Future Version Notes

1. The developer or user shall be provided mechanisms for adding new TAQs to the TAQ Table.

2. The developer or user shall not be allowed to delete TAQs from the TAQ Table.

3. The need for DELETE and DISREGARD tags for TAQs may be eliminated – DELETE and DISREGARD relations may both be replaced by a single matrix of relationships between

TAQs (as described in the Apply Surname TAQ Factors and Apply Given Name TAQ Factors sections of this document).

4. If DELETE and DISREGARD tags are maintained distinctly, the tool may also support DELETE and DISREGARD processing of Infixes, if necessary. Currently there are no infixes defined in the TAQ Table.

5. The tool may support conjoined TAQs.

   - Conjoined TAQs can be very effective in dealing with morphological endings.... (i.e., conjoined suffixes such as -son, -man, -ovich). Conjoined suffixes may be supported prior to cultural specific handling. Morphological endings trigger left bias right now, so we will need to consider left bias when implementing morphological lookups either as part of TAQ processing, or independent of it.

   - Conjoined TAQs will be even more effective if the tool supports culture-specific processing.

   - Once culture-specific processing is supported, the tool shall recognize that all TAQ types can be conjoined with a stem (i.e., prefix, suffix, infix, title, qualifier).

   - If a TAQ is identified as conjoined (i.e., the field SEPARATE-IF-CONJOINED = "T", then the tool shall consider this TAQ if it is conjoined to a stem as well as if it is a stand-alone name segment (i.e., the TAQ is surrounded by white space); if the tool is identified as not conjoined (i.e., the field SEPARATE-IF-CONJOINED = "F",), then the tool shall consider the TAQ if found as a stand-alone name segment. Thus, the SEPARATE-IF-CONJOINED field indicates whether the application program will search for a TAQ as an independent name segment as well as part of a name segment.

   - Conjoined processing shall determine whether a TAQ is conjoined either at the beginning or at the end of a name segment. Conjoined processing does <u>not</u> search for a TAQ anywhere within the name segment.

   - If the tool identifies a conjoined TAQ in a name segment; it shall:

     - create multiple segments by separating the TAQ(s) from the stem; and

     - then proceed with TAQ processing of the separated segments.

6. There is an outstanding issue for handling the apostrophe – this issue will not be resolved in Version 1.0. The issue is that in some cases, such as with a name like "O'Connor", we want to separate the conjoined "O' "from "Connor", and then recognize the "O' " as a TAQ and process it as is defined in the TAQ Table (i.e., DISREGARD or DELETE). In a case such as "Ol'ga", however, we simply want to delete the apostrophe to produce "Olga". For Version 1.0, we are defining the apostrophe as a removal marker and mapping its

occurrence to "NIL"; so "Oconnor" and "Olga" will be produced for the examples cited above. Once the tool supports conjoined TAQs, the TAQ "O" may be recognized and processed as a conjoined TAQ, which would produce the desired "Connor". The current TAQ Table has entries for "D' ", 'L' ", and "O' " as well as their counterparts, "D", "L", and "O" – all of which are marked SEPARATE-IF-CONJOINED. Further analysis is required to determine whether it is necessary to handle the apostrophe using a different technique(s). If we determine that the apostrophe will always (for all foreseeable future versions of the tool) be removed prior to TAQ processing, then the "D' ", 'L' ", and "O' " entries in the TAQ Table will no longer be necessary.

7. The tool may attempt to define the gender of a name based on the available TAQs.

8. In later versions of the tool, additional cultural partitions may be supported in the TAQ Table.

9. In the future, the tool may process TAQs differently based on culture (note that this is part of the justification for separating Generic from Anglo).

5.1.8 Identify number of segments in name fields

5.1.8.1 Functionality

After TAQ removal, the system shall identify the number of segments in the SN and GN fields to assist in producing the ordered list of the top X names.

At a minimum, the tool shall require one SN segment and one GN segment for each name.

The tool shall accept an empty string as a single SN segment or GN segment.

If no segment exists after TAQ removal, the tool shall create a single empty segment for the appropriate name field. Thus, the tool shall recognize a single empty segment to indicate no data after TAQ removal.

The tool shall tag all empty segments as "unknown".

The tool shall support up to 5 segments in both the SN and GN fields after removal of TAQs.

If more than 5 segments remain in a name field after TAQ removal, the additional segments will be excluded from the evaluation process.

5.1.9 Identify and process Given Name Variants (Query Name Only)

5.1.9.1 Functionality

5.1.9.1.1 GIVEN-NAME-VARIANT Table

The tool shall support a single GIVEN-NAME-VARIANT Table that describes the relationship between two Given Names based on a specified cultural perspective (GN-CULT-AFF-ID).

The GIVEN-NAME-VARIANT Table will consist of pairs of given names *within a culture* that are determined to be variants of one another, based on their having the same name stem. In other words, the type of variation defined for the contents of the GIVEN-NAME-VARIANT Table are determined based on a specific cultural perspective (e.g., using an english or anglo perspective, "ELENA" and "HELENA" are considered "similar but different" names", however, they are considered "predictable spelling variants" when the pair is defined using a Hispanic perspective).

The criteria for whether or not a pair of variants will be included in the GIVEN-NAME-VARIANT Table will be based on the following defined types of variation:

Variation Values

VARIATION TYPE	EXAMPLE	DEFAULT VALUE
*Spelling variant - predictable	SEAN - SHAWN	0.95
*Abbreviation	MARIA - MA	0.90
*Nickname	FRANCISCO - PACO	0.90
*Same root - morph difference	BUSTO - BUSTOS	0.85
Different culture (translation)	FRANCISCO - FRANCIS	0.85
*Related - unpredictable difference	BUSTO - BUSTONES	0.80
Truncation	FRANCISCO - FRANCISC	0.70
Misspelling	MARIA - MRAIA	0.70
Similar name; not same root	SALAM - SALIM	0.65
Gender	MARIA - MARIO	0.50

The items marked with a * are culture-specific variants:

- **Spelling variation** may or may not be taken care of by digraph matching (for the product it will probably handle most reasonable variation).
- **Abbreviations** and **nicknames** depend on the culture; many, if not most, can be taken care of with lists that can be improved over time by restricting the culture relationship.
- **Same root/morphological difference** is definitely culture-specific, since the root and morphological elements can only be identified within a system; many of the differences (if they are short) can be handled with digraphs.

- **Related/unpredictable difference** is also within a cultural system; these are not the same name, however. Much of these differences can be handled with digraphs, too.
- **Truncation** and **misspelling** can also be said to be culture-specific, since you have to know how it was spelled in the first place to know if it's misspelled or truncated. Depending on how these are identified (i.e., if we know for sure what they are a variant of), these should perhaps receive a high value (e.g., 0.90).

Similar name; not same root variants will be included in the GIVEN-NAME-VARIANT Table to enable the tool to override a potentially high digraph score by assigning a lower variant score, if desired. Thus, a name that might qualify as a digraph variant, but which we do not consider a variant of the related name pair, would be less likely to qualify as a variant (e.g., "MUHAMAD" and "MAHMUD" may occur in the Table with an associated GNV-SCORE set very low because their variation type = "similar but different" and we would prefer not to see them appear as variants of one another).

In Version 1.0 of the tool, the CULT-AFF-ID will include Generic, Anglo, Arabic, Chinese, Hispanic, Korean, and Russian, if applicable. The table below lists the possible CULT-AFF-ID values:

CULT-AFF-ID	CULTURAL-AFFINITY
A	Arabic
C	Chinese
E	Anglo
G	Generic
H	Hispanic
K	Korean
R	Russian

The GIVEN-NAME-VARIANT Table shall not be modifiable by the developer or user in Version 1.0.

A separate data base utility will be developed to generate code representing the contents of the GIVEN-NAME-VARIANT Table which is currently stored in MS Access.

The following is a sample of the contents of the GIVEN-NAME-VARIANT Table:

GIVEN-NAME	GN-VARIANT	GNV-SCORE	GN-CULT-AFF-ID
AARON	AHARON	0.95	G
AARON	ARN	0.65	G
ABRAHAM	ABE	0.9	G
ABRAHAM	ABRAM	0.65	G
ABRAHAM	AVRAHAM	0.95	G
ABRAHAM	AVROM	0.65	G
ADAM	ADAMO	0.85	G
ADAM	ADAN	0.85	G
ADRIAN	ADRIEN	0.95	G
AGNES	AGGIE	0.9	G
AGNES	AGNESE	0.85	G

AGNES	INES	0.85	G
AHMED	AHMAD	0.95	G
ALAN	AL	0.9	G

Each entry in the GIVEN-NAME-VARIANT Table shall represent a bilateral relationship, and therefore only one entry will be required to support these bilateral relations (e.g., there will only be one entry in the table to define a relationship between "ABRAHAM" and "ABE").

Each entry in the GIVEN-NAME-VARIANT Table will be assigned a GNV-SCORE, which is based on a type of variation. The variation type will not be included in the GIVEN-NAME-VARIANT Table.

The GIVEN-NAME-VARIANT Table will not include self-relationships (i.e., "ABRAHAM" "ABRAHAM" 0 is not in the Table).

5.1.9.1.2 Given Name Variant Processing

If GivenNameCheckVariant = "T", the tool shall perform the following:

- First, the tool shall look up each query GN segment to determine whether it is included in the GIVEN-NAME-VARIANT Table for the appropriate cultural perspective (GN-CULT-AFF-ID) as defined by the API-defined query type.
    - If the query GN segment is included in the subset of the GIVEN-NAME-VARIANT Table associated with the selected cultural perspective, the tool shall associate all of its known variants within that cultural perspective, and their variation score (GNV-SCORE) with the query GN segment for use in the evaluation process.
    - If the query GN segment is not included in the subset of the GIVEN-NAME-VARIANT Table associated with the selected cultural perspective, and the selected cultural perspective is not "Generic", then the tool shall look up each query GN segment to determine whether it is included in the "Generic" subset of the GIVEN-NAME-VARIANT Table.
        - If the query GN segment is included in the "Generic" subset of the GIVEN-NAME-VARIANT Table, the tool shall associate all of its known variants within the "Generic" subset, and their variation score (GNV-SCORE) with the query GN segment for use in the evaluation process.
        - If the query GN segment is not included in the "Generic" subset of the GIVEN-NAME-VARIANT Table, the tool shall perform no additional Given Name Variant processing for this GN segment.

5.1.9.2 Design Notes

1. In version 1.0, the Anglo contents of the GIVEN-NAME-VARIANT Table are derived from the 389 given names that occurred at least 200 times in the State Department Passport Database.

2. Hispanic Given Name Variants were gathered from the HNA variant table LAS generated for the State Department.

3. Korean Given Name Variants were gathered from the data LAS generated for ORD-C.

4. Chinese Given Name Variants were gathered from the DNC variant table LAS generated for the State Department.

5. Arabic Given Name Variants were gathered from the DNC variant table LAS generated for the State Department.

5.1.9.3 Future Version Notes

1. The developer or user shall be provided mechanisms for adding new variants to the GIVEN-NAME-VARIANT Table.

2. The developer or user shall not be allowed to delete variants from the GIVEN-NAME-VARIANT Table.

3. The tool may attempt to determine the gender of a name based on the Given Name variants – this may be especially valuable for Hispanic given names. In order to do this, the GIVEN-NAME-VARIANT Table would be enhanced to include gender information.

4. The GIVEN-NAME-VARIANT Table may support additional cultural perspectives.

5. In the future, the tool may process GN Variants differently based on culture (note that this is part of the justification for separating Generic from Anglo).

5.1.10 Identify and process Surname Variants (Query Name Only)

5.1.10.1 Functionality

5.1.10.1.1 SURNAME-VARIANT Table

The tool shall support a single SURNAME-VARIANT Table that describes the relationship between two Surnames based on a specified cultural perspective (SN-CULT-AFF-ID).

The SURNAME-VARIANT Table will consist of pairs of surnames *within a culture* that are determined to be variants of one another, based on their having the same name stem. In other words, the type of variation defined for the contents of the SURNAME-VARIANT Table are determined based on a specific cultural perspective.

The criteria for whether or not a pair of variants will be included in the SURNAME-VARIANT Table will be based on the following defined types of variation:

Variation Values

VARIATION TYPE	EXAMPLE	DEFAULT VALUE
*Spelling variant - predictable	GOMEZ - GOMES	0.95
*Abbreviation	GOMEZ - GOM	0.90
*Same root - morph difference	BUSTO - BUSTOS	0.85
*Related - unpredictable difference	BUSTO - BUSTONES	0.80
Truncation	FRANCISCO - FRANCISC	0.70
Misspelling	GOMEZ - GMEZ	0.70
Similar name; not same root	GOMEZ - GAMEZ	0.65

The items marked with a * are culture-specific variants:

- **Spelling variation** may or may not be taken care of by digraph matching (for the product it will probably handle most reasonable variation).
- **Abbreviations** and **nicknames** depend on the culture; many, if not most, can be taken care of with lists that can be improved over time by restricting the culture relationship.
- **Same root/morphological difference** is definitely culture-specific, since the root and morphological elements can only be identified within a system; many of the differences (if they are short) can be handled with digraphs.
- **Related/unpredictable difference** is also within a cultural system; these are not the same name, however. Much of these differences can be handled with digraphs, too.
- **Truncation** and **misspelling** can also be said to be culture-specific, since you have to know how it was spelled in the first place to know if it's misspelled or truncated. Depending on how these are identified (i.e., if we know for sure what they are a variant of), these should perhaps receive a high value (e.g., 0.90).

Variant Surnames based on morphological endings will only be included, if they will not be handled by other processing (i.e., conjoined TAQ (i.e., suffix) removal processing or by a morphological lookup table that will trigger the left bias factor). In other words, we are focusing on stem variations.

Similar name; not same root variants will be included in the SURNAME-VARIANT Table to enable the tool to override a potentially high digraph score by assigning a lower variant score, if desired. Thus, a name that might qualify as a digraph variant, but which we do not consider a variant of the related name pair, would be less likely to qualify as a variant.

The following additional types of variation will not be included in the SURNAME-VARIANT Table (even though they are included in the GIVEN-NAME-VARIANT Table):

- Nicknames;
- Different culture (translation); and
- Gender variants.

In Version 1.0 of the tool, the CULT-AFF-ID will include Generic, Anglo, Arabic, Chinese, Hispanic, Korean, and Russian, if applicable. The table below lists the possible CULT-AFF-ID values:

CULT-AFF-ID	CULTURAL AFFINITY
A	Arabic
C	Chinese
E	Anglo
G	Generic
H	Hispanic
K	Korean
R	Russian

The SURNAME-VARIANT Table shall not be modifiable by the developer or user in Version 1.0.

A separate data base utility will be developed to generate code representing the contents of the SURNAME-VARIANT Table which is currently stored in MS Access.

The following is a sample of the contents of the SURNAME-VARIANT Table:

SURNAME	SN-VARIANT	SNV-SCORE	SN-CULT-AFF-ID
ACOSTA	COSTA	0.85	H
AGUILAR	AGUILA	0.65	H
AGUILAR	AGUILERA	0.65	H
AGUILERA	AGUILA	0.85	H
AGUILERA	AGUILAR	0.65	H
ALBA	ALBAN	0.65	H
ALCANTARA	ALCANTAR	0.85	H
ALDANA	ALDAMA	0.8	H
ALMANZAR	ALMANZA	0.65	H
ALONSO	ALONZO	0.95	H
ALONZO	ALONSO	0.95	H
ALVARADO	ALVARDO	0.8	H
ALVAREZ	ALVARES	0.95	H
ALVAREZ	ALVARO	0.65	H
ALVAREZ	ALVEREZ	0.8	H

Each entry in the SURNAME-VARIANT Table shall represent a bilateral relationship, and therefore only one entry will be required to support these bilateral relations (e.g., there will only be one entry in the table to define a relationship between "GOMEZ" and "GOMES").

Each entry in the SURNAME-VARIANT Table will be assigned a SNV-SCORE, which is based on a type of variation. The variation type will not be included in the SURNAME -VARIANT Table.

The SURNAME-VARIANT Table will not include self-relationships (i.e., "GOMEZ" "GOMEZ" 0 is not in the table).

5.1.10.1.2 Surname Variant Processing

If SurnameCheckVariant = "T", the tool shall perform the following:

- First, the tool shall look up each query SN segment to determine whether it is included in the SURNAME-VARIANT Table for the appropriate cultural perspective (SN-CULT-AFF-ID), as defined by the API-defined query type.
    - If the query SN segment is included in the subset of the SURNAME-VARIANT Table associated with the selected cultural perspective the tool shall associate all of its known variants within that cultural perspective, and their variation score (SNV-SCORE) with the query SN segment for use in the evaluation process.
    - If the query SN segment is not included in the subset of the SURNAME-VARIANT Table associated with the selected cultural perspective, and the selected cultural perspective is not "Generic", then the tool shall look up each query SN segment to determine whether it is included in the "Generic" subset of the SURNAME-VARIANT Table.
        - If the query SN segment is included in the "Generic" subset of the SURNAME-VARIANT Table, the tool shall associate all of its known variants within the "Generic" subset, and their variation score (SNV-SCORE) with the query SN segment for use in the evaluation process.
        - If the query SN segment is not included in the "Generic" subset of the SURNAME-VARIANT Table, the tool shall perform no additional Surname Variant processing for this SN segment.

5.1.10.2 Design Notes

1. The type of variation defined for the Anglo contents of the SURNAME-VARIANT Table were determined based on different cultural perspectives using the phonetic workbench to generate variants that would not be handled by a digraph search.
2. Hispanic Surname Variants were gathered from the HNA variant table LAS generated for the State Department.

3. Korean Surname Variants were gathered from the data LAS generated for ORD-C and supplemented with entries in the DNC variant table LAS generated for the State Department.

4. Chinese Surname Variants were gathered from the DNC variant table LAS generated for the State Department.

5.1.10.3 Future Version Notes

1. The developer or user shall be provided mechanisms for adding new variants to the SURNAME-VARIANT Table.

2. The developer or user shall not be allowed to delete variants from the SURNAME-VARIANT Table.

3. The SURNAME-VARIANT Table may support additional cultural perspectives.

4. In the future, the tool may process SN Variants differently based on culture (note that this is part of the justification for separating Generic from Anglo).

6. Evaluate and Score

6.1 Functionality

The tool shall compare each candidate name with the query name to determine whether the candidate name qualifies as a similar name.

In order to determine whether the candidate name is similar to the query name, the tool shall:

- **Evaluate the Surname;**
- **Evaluate the Given Name;**
- **Determine if the SurnameScore exceeds SurnameThreshold;**
- **Determine if the GivenNameScore exceeds GivenNameThreshold;** and
- then **Compute a NameScore & Determine If Potential Match.**

6.1.1 Evaluate Surname

In order to evaluate the Surname, the tool shall:

- First **Determine a SurnameSegmentScore** for each possible pairing of query and candidate SN segments;

- Then **Apply SN Segment Evaluation Factors** to adjust the SurnameSegmentScore (resulting from either a Surname Variant match, Surname Initial match, not exist or unknown match, or a Surname Digraph match) by multiplying the SurnameSegmentScore according to a set of Surname evaluation factors; and

- Finally, **Determine a SurnameScore.**

6.1.1.1 Determine SurnameSegmentScore

The tool shall compare each of the candidate SN segments with the query SN segments to determine a SurnameSegmentScore for each pair of SN segments.

This pairing of SN segments can be represented in an evaluation matrix, such as the one depicted below.

	Granier	Smith
Smyth	SurnameSegmentScore1	SurnameSegmentScore2

SurnameSegmentScore1 = SurnameSegmentScore determined when comparing "Smith" and "Granier".
SurnameSegmentScore2 = SurnameSegmentScore * Evaluation Factor determined when comparing "Smith" and "Smith".

The tool shall determine each SurnameSegmentScore as follows:

- First **Check for Not Exist or Unknown Values (SurnameCheckUnknownNotExist, LastNameUnknownScore, NoLastNameScore)** on the two SN segments.

- If the two SN segments are not a Not Exist or Unknown match, **Check for Surname Variant Match (SurnameCheckVariant, SNV-SCORE)**

- If the two SN segments are not a Not Exist or Unknown match or Surname Variant match, **Check for Surname Initial Match (SurnameCheckInitial, SurnameInitialScore, SurnameExactInitialMatchScore)**

- If the two SN segments are not a Not Exist or Unknown match, or a Surname Variant match, or a Surname Initial match, then the tool shall **Perform a Surname Digraph Comparison (SurnameCheckBias)**

6.1.1.1.1 Check for Not Exist or Unknown Values (SurnameCheckUnknownNotExist, LastNameUnknownScore, NoLastNameScore)

If SurnameCheckUnknownNotExist = "T", then the tool shall determine whether the NoLastNameScore or LastNameUnknownScore can be assigned to the SurnameSegmentScore to handle a SN segment that does not exist or whose value is unknown.

The following table illustrates the SurnameCheckUnknownNotExist conditions and associated values for setting the SurnameSegmentScore:

comparand A	**comparand B known**	**comparand B unknown**	**comparand B not exist**
known	N/A	LastNameUnknownScore	NoLastNameScore
unknown	LastNameUnknownScore	(LastNameUnknownScore + 1)/2	(LastNameUnknownScore +1)/2
not exist	NoLastNameScore	(LastNameUnknownScore +1)/2	(NoLastNameScore + 1)/2

If one comparand is defined as "unknown" and the other comparand is "known", then the tool shall set the SurnameSegmentScore = LastNameUnknownScore.

If one comparand is identified as "unknown", and the other comparand is defined as "not exist", then the tool shall set the SurnameSegmentScore = (LastNameUnknownScore+1)/2.

If one comparand is defined as "known" and the other comparand is "not exist", then the tool shall set the SurnameSegmentScore = NoLastNameScore.

If both comparands are identified as "unknown", then the tool shall set the SurnameSegmentScore = (LastNameUnknownScore+1)/2.

If both comparands are identified as "not exist", then the tool shall set the SurnameSegmentScore = (NoLastNameScore+1)/2.

6.1.1.1.2 Check for Surname Variant Match (SurnameCheckVariant, SNV-SCORE)

If SurnameCheckVariant = "T", the tool shall determine whether a SNV-SCORE can be applied.

For every segment pairing, the tool shall determine whether the two SN segments have been predetermined to be variants of one another (i.e., defined in the SURNAME-VARIANT Table) by checking to see if the candidate SN segment is present in the list of variants associated with the query SN segment.

If the candidate SN segment is present in the list of variants associated with the query SN segment, then the tool shall set the SurnameSegmentScore = SNV-SCORE associated with the query variant.

6.1.1.1.3 Check for Surname Initial Match (SurnameCheckInitial, SurnameInitialScore, SurnameExactInitialMatchScore)

If SurnameCheckInitial = "T" and the SN segment was not identified as a Surname Variant, then the tool shall determine whether the SurnameInitialScore or SurnameExactInitialMatchScore can be applied.

If comparand A's SN segment is a single character and comparand B's SN segment is a single character and they match, then the tool shall set the SurnameSegmentScore = SurnameExactInitialMatchScore.

If comparand A's SN segment is a single character and comparand B's SN segment is more than one character and comparand A's SN segment matches the first character of comparand B's SN segment, the tool shall set the SurnameSegmentScore = SurnameInitialScore.

6.1.1.1.4 Perform a Surname Digraph Evaluation

A value from 0.0 to 1.0 shall be calculated based on the number of digraphs which match between two SN segments.

A digraph shall only participate in a match once.

One point shall be awarded for each digraph that participates in a match, thus each digraph match shall result in exactly two points being added to the total digraph score.

The SurnameSegmentScore shall be the total number of points assigned based on the matching digraphs (digraph score), divided by the number of digraphs that occur in the two SN segments.

For example, the SN segments "Garcia" and "Garica" are not an exact match. Of fourteen total digraphs involved in the evaluation, there are four matches, involving 8 digraphs.

Query SN Segment: Garcia
Candidate SN Segment : Garica

#G Ga ar rc ci ia a#
#G Ga ac cr ri ia a#
#G Ga ia a#

Therefore, the name receives a digraph score of 8/14 = .57

6.1.1.1.4.1 Apply Surname Left Digraph Bias (SurnameCheckBias)

If SurnameCheckBias = "T", then a bias will be applied so that digraphs on the right end of the strings count less than those on the left in a particular SN segment.

If SurnameCheckBias = "T", the tool shall apply bias by:

- First, assigning the first contributing digraph in each SN segment a weight factor of 1.00, the second contributing digraph a weight factor of .9, and so on until the tenth contributing digraph is reached, at which point, all remaining digraphs shall be assigned a weight factor of .1.
- then determine which digraphs match between the two SN segments;
- sum the weight factors assigned to the matched query SN digraphs with the weight factors assigned to the matched candidate SN digraphs; and
- divide by the sum of all contributing digraphs for both the query SN and the candidate SN.

SurnameCheckBias : T

Query : Moskyovich, FNU
Candidate : Markovich, FNU

-M Mo os sk ky yo ov vi ic ch h-
(1.0) (.9) (.8)(.7)(.6) (.5) (.4)(.3)(.2) (.1)(.1)

-M Ma ar rk ko ov vi ic ch h-
(1.0) (.9) (.8)(.7)(.6) (.5)(.4)(.3)(.2) (.1)

The following digraphs match:

	-M	ov	vi	ic	ch	h-
Query Weight Factors:	(1.0)	(.4)	(.3)	(.2)	(.1)	(.1)
Candidate Weight Factors:	(1.0)	(.5)	(.4)	(.3)	(.2)	(.1)

$$\frac{\text{Matched Digraphs}}{\text{Total possible Digraphs}} = \frac{(1.0+.4+.3+.2+.1+.1)+(1.0+.5+.4+.3+.2+.1)}{(1.0+.9+.8+.7+.6+.5+.4+.3+.2+.1+.1)+(1.0+.9+.8+.7+.6+.5+.4+.3+.2+.1)}$$

Therefore, the SurnameSegmentScore == (4.6 / 11.1) = 0.41

6.1.1.1.5 Design Notes

1. We may want to support a Right Bias in the future.

6.1.1.2 Apply SN Segment Evaluation Factors

SN Segment Evaluate Factors will be applied, if appropriate, to each segment in the evaluation matrix.

The tool shall determine whether to apply certain SN Segment Evaluation Factors in determining a similar name, based on the application of the following set of logical parameters:

- **Determine Relative Position of SN Segments (SurnameAnchorSegment);**
- **Apply Surname Out of Position Factor (SurnameOutOfPositionFactor);**
- **Apply Surname Anchor Segment Factor (SurnameAnchorSegment, SurnameAnchorFactor)**; and
- **Apply Surname TAQ Factors (SurnameCheckTAQ, SurnameTAQDeleteFactor, SurnameTAQDisregardFactor, SurnameTAQDisregardAbsentFactor, SurnameTAQDeleteAbsentFactor).**

6.1.1.2.1 Determine Relative Position of SN Segments (SurnameAnchorSegment)

In order to determine the relative position in both comparands, the tool shall establish an "index" of a segment based on the SurnameAnchorSegment.

For SurnameAnchorSegment = "none" or "first", the tool shall count segments from left to right.

For SurnameAnchorSegment = "last", the tool shall count segments from right to left.

6.1.1.2.2 Apply Surname Out of Position Factor (SurnameOutOfPositionFactor)

If a SN segment is out of position, the SurnameOutOfPositionFactor will always be applied.

If two SN segments are not in the same relative position in both comparands, the tool shall multiply the SurnameSegmentScore by the SurnameOutOfPositionFactor.

In the example cited below, "Smyth" and "Smith" are out of position, and thus the SurnameOutOfPosition factor will be applied to SurnameSegmentScore2.

	Granier	Smith
Smyth	SurnameSegmentScore1	SurnameSegmentScore2

6.1.1.2.3 Apply Surname Anchor Segment Factor (SurnameAnchorFactor, SurnameAnchorSegment)

The SurnameAnchorFactor is used to identify and emphasize the importance of one segment of the surname over another if more than one SN segment exists.

The SurnameAnchorFactor shall never be applied if the SurnameMode is "average" as this would doubly discount the contribution of a single SN segment when determining an overall SN score.

The SurnameAnchorFactor shall never be applied if the SurnameOutOfPositionFactor has already been applied, even if the SurnameAnchorSegment = "first" or "last". Thus, the SN segments must be in position if SurnameAnchorFactor will be applied.

When the SurnameAnchorSegment = "none", neither SN segment will be assigned more weight than the other (i.e., Anchor segment is essentially turned off). When the SurnameAnchorSegment = "first", the first (i.e., left-most) SN segment will be assigned more weight, and when the SurnameAnchorSegment = "last", the last (i.e., right-most) SN segment will be assigned more weight.

If two SN segments are in the same relative position in both comparands (i.e., SurnameOutOfPositionFactor did not apply), and SurnameMode is "lowest" or "highest", and their position is not the SurnameAnchorSegment, and SurnameAnchorSegment = "first" or "last", then the tool shall multiply the SurnameSegmentScore by the SurnameAnchorFactor.

6.1.1.2.4 Apply Surname TAQ Factors (SurnameCheckTAQ, SurnameTAQDeleteFactor, SurnameTAQDisregardFactor, SurnameTAQDisregardAbsentFactor, SurnameTAQDeleteAbsentFactor)

6.1.1.2.4.1 Functionality

DISREGARD TAQs are viewed during evaluation as more important than DELETE TAQs. Two TAQs of the same type (i.e., DELETE or DISREGARD) that do not match are viewed during evaluation as more important than absence of a TAQ type in one comparand and presence of that same TAQ type in the other comparand.

When the SurnameCheckTAQ = "off" or "remove", no TAQ processing shall take place during the evaluation process.

When the SurnameCheckTAQ = "score", TAQs that were identified, removed, and associated with each relevant SN segment during preprocessing shall be factored into the SurnameSegmentScore.

If SurnameCheckTAQ = "score", the tool shall determine which of the following four parameters can be applied to the SurnameSegmentScore:

- **SurnameTAQDisregardFactor;**
- **SurnameTAQDeleteFactor**;
- **SurnameTAQDisregardAbsentFactor**; and
- **SurnameTAQDeleteAbsentFactor**.

TAQ processing shall be performed as follows:

- First, determine whether the query name segment and the candidate name segment each have associated TAQs identified during pre-processing.
- If no TAQs are associated with either segment, then the tool shall not adjust the SurnameSegmentScore (i.e., DELETE TAQs None Occur, DISREGARD TAQs None Occur).
- If one comparand has all DELETE TAQs associated with it and the other comparand has no TAQs associated with it, then the SurnameSegmentScore will be multiplied by the SurnameTAQDeleteAbsentFactor (i.e., DELETE TAQs Absent, DISREGARD TAQs None Occur).
- If one or more DISREGARD TAQs are associated with one comparand and not the other (i.e., either the query name segment or the candidate name segment has one or more DISREGARD TAQs), then the tool shall apply the SurnameTAQDisregardAbsentFactor (i.e., DELETE TAQs >=1 Match, No Match, Absent, None Occur, DISREGARD TAQs Absent).
- If DISREGARD TAQs are present in both comparands, then the tool shall determine whether there are any matches on any of the DISREGARD TAQs:
  - If any matches are found, then the tool shall determine if there are any matches on any DELETE TAQs.
    - If no DELETE TAQs are present, then the tool shall not adjust the SurnameSegmentScore (i.e., DELETE TAQs None Occur, DISREGARD TAQs >=1 Match).
    - If DELETE TAQs are present in both comparands and no matches are found, then the SurnameSegmentScore will be multiplied by the SurnameTAQDeleteFactor (i.e., DELETE TAQs No Match, DISREGARD TAQs >=1 Match).
    - If DELETE TAQs are present in both comparands and a match is found, then the tool shall not adjust the SurnameSegmentScore (i.e., DELETE TAQs >=1 Match, DISREGARD TAQs >=1 Match).
    - If DELETE TAQs are present in one comparand, but not the other, then the SurnameSegmentScore will be multiplied by the SurnameTAQDeleteAbsentFactor (i.e., DELETE TAQs Absent, DISREGARD TAQs >=1 Match).
  - If no match is found, then the SurnameSegmentScore will be multiplied by the SurnameTAQDisregardFactor(i.e., DELETE TAQs >=1 Match, No Match, Absent, None Occur, DISREGARD TAQs No Match).

- If both comparands have all DELETE TAQs associated with them, the tool shall determine if any of the DELETE TAQs match:
    - If there is any match, then the tool shall not adjust the SurnameSegmentScore (i.e., DELETE TAQs >=1 Match DISREGARD TAQs None Occur).
    - If there is no match, then the SurnameSegmentScore will be multiplied by the SurnameTAQDeleteFactor (i.e., DELETE TAQs No Match DISREGARD TAQs None Occur).

The following table describes the conditions governing the application of TAQ parameters as described in the text above:

DELETE TAQ(s)	DISREGARD TAQ(s)	Impact on SurnameSegmentScore
None Occur	None Occur	No Change
None Occur	No Match	Apply SurnameTAQDisregardFactor
None Occur	Absent	Apply SurnameTAQDisregardAbsentFactor
None Occur	>=1 Match	No Change
No Match	None Occur	Apply SurnameTAQDeleteFactor
No Match	No Match	Apply SurnameTAQDisregardFactor
No Match	Absent	Apply SurnameTAQDisregardAbsentFactor
No Match	>=1 Match	Apply SurnameTAQDeleteFactor
Absent	None Occur	Apply SurnameTAQDeleteAbsentFactor
Absent	No Match	Apply SurnameTAQDisregardFactor
Absent	Absent	Apply SurnameTAQDisregardAbsentFactor
Absent	>=1 Match	Apply SurnameTAQDeleteAbsentFactor
>=1 Match	None Occur	No Change
>=1 Match	No Match	Apply SurnameTAQDisregardFactor
>=1 Match	Absent	Apply SurnameTAQDisregardAbsentFactor
>=1 Match	>= 1 Match	No Change

"Match" in the table indicates that the stated TAQ Type occurs in both comparands and at least one of the TAQ Type values occurs in both comparands, i.e., the TAQ Type values are the same. For example, the DELETE TAQ value "Mr" may occur in both comparands.

"No Match" in the table indicates that the stated TAQ Type occurs in both comparands but none of the TAQ Type values occurs in both comparands, i.e., the values are not the same. For example, the single DELETE TAQ value "Mr" may occur in one comparand and the single DELETE TAQ value "Mrs" may occur in the other comparand.

"Absent" in the table indicates that the stated TAQ Type is absent in one of the comparands but occurs in the other comparand. For example, the DELETE TAQ value "Mr" may occur in one comparand and there may be no DELETE TAQ value at all in the other comparand.

"None Occur" in the table indicates that the stated TAQ Type does not occur in either comparand. For example, no DELETE TAQs occur in either comparand.

6.1.1.2.4.2 Future Version Notes

- The SurnameTAQDisregardFactor will be replaced by the highest TAQ-DISREGARD-WEIGHT for each DISREGARD TAQ relationship that is defined in a TAQ-DISREGARD-WEIGHT Table. The TAQ-DISREGARD-WEIGHT Table defines a weighted relationship between two TAQs that occur within a specified cultural boundary or partition. There may be a default TAQ-DISREGARD-WEIGHT established so that only those TAQ relationships that warrant special weighting may be entered into the TAQ-DISREGARD-WEIGHT Table.

- When the TAQ-DISREGARD-WEIGHT Table is implemented, the importance of DISREGARD versus DELETE TAQs may change. For example, MR and MRS are currently defined as DELETE TAQs, and therefore considered less important than other TAQ values. However, their relationship to one another may be treated with more significance in later versions due to gender specification.

6.1.1.3 Determine SurnameScore

In order to determine the SurnameScore, the tool shall perform the following:

- **Compute the Highest SurnameSegmentScore(s) (SurnameMode="Highest")**
- **Compute the Best Combination of SurnameSegmentScore(s) (SurnameMode="Average")**
- **Compute the Lowest SurnameSegmentScore(s) (SurnameMode="Lowest")**

If either comparand has just one SN segment, then the tool shall set the SurnameScore = the Highest (Best) SurnameSegmentScore found in the evaluation matrix.

If more than one segment occurs in both surnames, then the tool shall **Apply the Surname Mode** in its determination of the SurnameScore.

6.1.1.3.1 Compute Highest SurnameSegmentScore(s) (SurnameMode="Highest")

The tool shall compute the highest set of SurnameSegmentScores (includes the highest SurnameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that includes the highest set of SurnameSegmentScores.

In the following example, the highest SurnameSegmentScores will be 1.0 and .57, since 1.0 is the highest SurnameSegmentScore.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

In the following example, the highest SurnameSegmentScores will be 1.0 and .62, since 1.0 is the highest SurnameSegmentScore, and .62 is the next highest SurnameSegmentScore that is in a different row and column combination in the matrix.

	Garcia	Garza	Garza

Garica	.57	.62	.62
Garza	.62	1.0	1.0

6.1.1.3.2 Compute Best Combination of SurnameSegmentScore(s) (SurnameMode="Average")

The tool shall compute the best possible combination of scores (i.e., the Highest Average of SurnameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that gives the highest sum.

In the following example, the best combination of SurnameSegmentScores will be 1.0 and .57, since ((1.0+.57)/2) > ((.62+.62)/2) = .79 > .62.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

6.1.1.3.3 Compute Lowest SurnameSegmentScore(s) (SurnameMode="Lowest")

The tool shall compute the lowest set of SurnameSegmentScores (includes the Lowest SurnameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that includes the lowest set of SurnameSegmentScores.

In the following example, the lowest SurnameSegmentScores will be .57 and 1.0, since .57 is the lowest SurnameSegmentScore.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

In the following example, the lowest SurnameSegmentScores will be .57 and 1.0, since .57 is the lowest SurnameSegmentScore, and 1.0 is the next lowest SurnameSegmentScore that is in a different row and column combination in the matrix.

	Garcia	Garza	Garza
Garica	.57	.62	.62

Garza	.62	1.0	1.0

6.1.1.3.4 Apply Surname Mode (SurnameMode)

If SurnameMode = "highest", the tool shall set the SurnameScore = the highest SurnameSegmentScore found in the evaluation matrix.

If SurnameMode = "average", the tool shall set the SurnameScore = the average of the SurnameSegmentScores found in the evaluation matrix..

If SurnameMode = "lowest", the tool shall set the SurnameScore = the lowest SurnameSegmentScore found in the evaluation matrix.

6.1.1.3.5 Determine SurnameCompressedScore (SurnameCheckCompressed, SurnameCompressedScore)

This function handles names that are essentially the same name but are segmented differently (e.g., "de la Garcia" → "delaGarcia").

If SurnameCheckCompressed = "T", then the tool shall generate a SurnameCompressedScore in the following manner:

- Create query and candidate Compressed SN fields from their original SN fields, by processing segmentation and removal markers, and then eliminating all remaining blanks.
- Compare the query and candidate COMPRESSED SN fields to determine if there is an exact match.
- If there is an exact match of the COMPRESSED SN fields, the tool shall set the SurnameScore to the higher score of the SurnameCompressedScore or the previously calculated SurnameScore.

6.1.2 Evaluate Given Name

In order to evaluate the Given Name, the tool shall:

- First **Determine a GivenNameSegmentScore** for each possible combination of query and candidate GN segments;
- Then, **Apply GN Segment Evaluation Factors** to adjust the GivenNameSegmentScore (resulting from either a Given Name Variant match, Given Name Initial match, not exist or unknown match, or a Given Name Digraph match) by multiplying the GivenNameSegmentScore according to a set of Given Name evaluation factors.

- Finally, **Determine a GivenNameScore**.

6.1.2.1 Determine GivenNameSegmentScore

The tool shall compare each of the candidate GN segments with the query GN segments to determine a GivenNameSegmentScore for each pair of GN segments.

This pairing of GN segments can be represented in an evaluation matrix similar to the one described in the section Determine SurnameSegmentScore.

The tool shall determine each GivenNameSegmentScore as follows:

- First **Check for Not Exist or Unknown Values (GivenNameCheckUnknownNotExist, FirstNameUnknownScore, NoFirstNameScore)** on the two GN segments.
- If the two GN segments are not a Not Exist or Unknown match, **Check for Given Name Variant Match (GivenNameCheckVariant, GNV-SCORE)**
- If the two GN segments are not a Not Exist or Unknown match or Given Name Variant match, **Check for Given Name Initial Match (GivenNameCheckInitial, GivenNameInitialScore, GivenNameExactInitialMatchScore)**
- If the two GN segments are not a Not Exist or Unknown match, or a Given Name Variant match, or a Given Name Initial match, then the tool shall **Perform a Given Name Digraph Comparison (GivenNameCheckBias)**

6.1.2.1.1 Check for Not Exist or Unknown Values (GivenNameCheckUnknownNotExist, FirstNameUnknownScore, NoFirstNameScore)

If GivenNameCheckUnknownNotExist = "T", then the tool shall determine whether the NoFirstNameScore or FirstNameUnknownScore can be assigned to the GivenNameSegmentScore to handle a GN segment that does not exist or whose value is unknown.

The following table illustrates the GivenNameCheckUnknownNotExist conditions and associated values for setting the GivenNameSegmentScore:

comparand.A	**comparand B known**	**comparand B unknown**	**comparand B not exist**
known	N/A	FirstNameUnknownScore	NoFirstNameScore
unknown	FIrstNameUnknownScore	(FirstNameUnknownScore + 1)/2	(FirstNameUnknownScore +1)/2
not exist	NoFirstNameScore	(FirstNameUnknownScore +1)/2	(NoFirstNameScore + 1)/2

If one comparand is defined as "unknown" and the other comparand is "known", then the tool shall set the GivenNameSegmentScore = FirstNameUnknownScore.

If one comparand is defined as "known" and the other comparand is "not exist", then the tool shall set the GivenNameSegmentScore = NoFirstNameScore.

If both comparands are identified as "not exist" or both are identified as "unknown", then the tool shall set the GivenNameSegmentScore = (FirstNameUnknownScore + 1)/2.

If both comparands are identified as either "not exist" or "unknown", and the comparands are not defined the same, then the tool shall set the GivenNameSegmentScore = (FirstNameUnknownScore+1)/2.

If both comparands are identified as "unknown", then the tool shall set the GivenNameSegmentScore = (FirstNameUnknownScore+1)/2.

If both comparands are identified as "not exist", then the tool shall set the GivenNameSegmentScore = (NoFirstNameScore+1)/2.

6.1.2.1.2 Check for Given Name Variant Match (GivenNameCheckVariant, GNV-SCORE)

If GivenNameCheckVariant = "T", the tool shall determine whether the GNV-SCORE can be applied.

For every segment pairing, the tool shall determine whether the two GN segments have been predetermined to be variants of one another (i.e., defined in the GIVEN-NAME-VARIANT Table) by checking to see if the candidate GN segment is present in the list of variants associated with the query GN segment.

If the candidate GN segment is present in the list of variants associated with the query GN segment, then the tool shall set the GivenNameSegmentScore = GNV-SCORE associated with the query variant.

6.1.2.1.3 Check for Given Name Initial Match (GivenNameCheckInitial, GivenNameInitialScore, GivenNameExactInitialMatchScore)

If GivenNameCheckInitial = "T" and the GN segment was not identified as a Given Name Variant, then the tool shall determine whether the GivenNameInitialScore or GivenNameExactInitialMatchScore can be applied.

> If comparand A's GN segment is a single character and comparand B's GN segment is a single character and they match, then the tool shall set the GivenNameSegmentScore = GivenNameExactInitialMatchScore.
>
> If comparand A's GN segment is a single character and comparand B's GN segment is more than one character and comparand A's GN segment matches the first character of comparand B's GN segment, the tool shall set the GivenNameSegmentScore = GivenNameInitialScore.

6.1.2.1.4 Perform Given Name Digraph Evaluation

A value from 0.0 to 1.0 shall be calculated based on the number of digraphs which match between two Given Name segments.

A digraph shall only participate in a match once.

One point shall be awarded for each digraph that participates in a match, thus each digraph match shall result in exactly two points being added to the total digraph score.

The GivenNameSegmentScore shall be the total number of points assigned based on the matching digraphs (digraph score), divided by the number of digraphs that occur in the two GN segments.

6.1.2.1.4.1 Apply Given Name Left Digraph Bias (GivenNameCheckBias)

If GivenNameCheckBias = "T", then a bias will be applied so that digraphs on the right end of the strings count less than those on the left in a particular GN segment.

If GivenNameCheckBias = "T", the tool shall apply bias by:

- First, assigning the first contributing digraph in each GN segment a weight factor of 1.00, the second contributing digraph a weight factor of .9, and so on until the tenth contributing digraph is reached, at which point, all remaining digraphs shall be assigned a weight factor of .1.
- then determine which digraphs match between the two GN segments;
- sum the weight factors assigned to the matched query GN digraphs with the weight factors assigned to the matched candidate GN digraphs; and
- divide by the sum of all contributing digraphs for both the query GN and the candidate GN.

6.1.2.1.5 Design Notes

1. We may want to support a Right Bias in the future.

6.1.2.2 Apply GN Segment Evaluation Factors

The tool shall determine whether to apply certain GN Segment Evaluation Factors in determining a similar name, based on the application of the following set of logical parameters:

- **Determine Relative Position of GN Segments (GivenNameAnchorSegment);**
- **Apply Given Name Out of Position Factor (GivenNameOutOfPositionFactor);**

- **Apply Given Name Anchor Segment Factor (GivenNameAnchorSegment, GivenNameAnchorFactor);** and
- **Apply Given Name TAQ Factors (GivenNameCheckTAQ, GivenNameTAQDeleteFactor, GivenNameTAQDisregardFactor, GivenNameTAQDisregardAbsentFactor, GivenNameTAQDeleteAbsentFactor).**

6.1.2.2.1 Determine Relative Position of GN Segments (GivenNameAnchorSegment)

In order to determine the relative position in both comparands, the tool shall establish an "index" of a segment based on the GivenNameAnchorSegment.

For GivenNameAnchorSegment = "none" or "first", the tool shall count segments from left to right.

For GivenNameAnchorSegment = "last", the tool shall count segments from right to left.

6.1.2.2.2 Apply Given Name Out of Position Factor (GivenNameOutOfPositionFactor)

If a GN segment is out of position, the GivenNameOutOfPositionFactor will always be applied.

If two GN segments are not in the same relative position in both comparands, the tool shall multiply the GivenNameSegmentScore by the GivenNameOutOfPositionFactor.

In the example cited below, "Jeffrey" and "Jeffrey" are out of position, and thus the SurnameOutOfPosition factor will be applied to SurnameSegmentScore3.

	Jeffrey	Andrew
A	SurnameSegmentScore1	SurnameSegmentScore2
Jeffrey	SurnameSegmentScore3	SurnameSegmentScore4

6.1.2.2.3 Apply Given Name Anchor Segment Factor (GivenNameAnchorFactor, GivenNameAnchorSegment)

The GivenNameAnchorFactor is used to identify and emphasize the importance of one segment of the given name over another if more than one GN segment exists.

The GivenNameAnchorFactor shall never be applied if the GivenNameMode is "average" as this would doubly discount the contribution of a single GN segment when determining an overall GN score.

The GivenNameAnchorFactor shall never be applied if the GivenNameOutOfPositionFactor has already been applied, even if the GivenNameAnchorSegment = "first" or "last". Thus, the GN segments must be in position if GivenNameAnchorFactor will be applied.

When the GivenNameAnchorSegment = "none", neither GN segment will be assigned more weight than the other (i.e., Anchor segment is essentially turned off). When the GivenNameAnchorSegment = "first", the first (i.e., left-most) GN segment will be assigned more weight, and when the GivenNameAnchorSegment = "last", the last (i.e., right-most) GN segment will be assigned more weight.

If two GN segments are in the same relative position in both comparands (i.e., GivenNameOutOfPositionFactor did not apply), and their position is not the GivenNameAnchorSegment, and GivenNameAnchorSegment = "first" or "last", and GivenNameMode is "lowest" or "highest", then the tool shall multiply the GivenNameSegmentScore by the GivenNameAnchorFactor.

6.1.2.2.4 Apply Given Name TAQ Factors (GivenNameCheckTAQ, GivenNameTAQDeleteFactor, GivenNameTAQDisregardFactor, GivenNameTAQDisregardAbsentFactor, GivenNameTAQDeleteAbsentFactor)

6.1.2.2.4.1 Functionality

DISREGARD TAQs are viewed as more important than DELETE TAQs. Two TAQs of the same type (i.e., DELETE or DISREGARD) that do not match are viewed as more important than absence of a TAQ type in one comparand and presence of that same TAQ type in the other comparand.

When the GivenNameCheckTAQ = "off" or "remove", no TAQ processing will take place during the evaluation process.

When the GivenNameCheckTAQ = "score", TAQs that were identified, removed, and associated with each relevant GN segment during preprocessing will be factored into the GivenNameSegmentScore.

If GivenNameCheckTAQ = "score", the tool shall determine which of the following four parameters can be applied to the GIvenNameSegmentScore:

- **GivenNameTAQDisregardFactor;**
- **GivenNameTAQDeleteFactor**;
- **GivenNameTAQDisregardAbsentFactor**; and
- **GivenNameTAQDeleteAbsentFactor**.

TAQ processing shall be performed as follows:

- First, determine whether the query name segment and the candidate name segment each have associated TAQs identified during pre-processing.

- If no TAQs are associated with either segment, then the tool shall not adjust the GlvenNameSegmentScore (i.e., DELETE TAQs None Occur, DISREGARD TAQs None Occur).
- If one comparand has all DELETE TAQs associated with it and the other comparand has no TAQs associated with it, then the GivenNameSegmentScore will be multiplied by the GivenNameTAQDeleteAbsentFactor (i.e., DELETE TAQs Absent, DISREGARD TAQs None Occur).
- If one or more DISREGARD TAQs are associated with one comparand and not the other (i.e., either the query name segment or the candidate name segment has one or more DISREGARD TAQs), then the tool shall apply the GivenNameTAQDisregardAbsentFactor (i.e., DELETE TAQs >=1 Match, No Match, Absent, None Occur, DISREGARD TAQs Absent).
- If DISREGARD TAQs are present in both comparands, then the tool shall determine whether there are any matches on any of the DISREGARD TAQs:
    - If any matches are found, then the tool shall determine if there are any matches on any DELETE TAQs.
        - If no DELETE TAQs are present, then the tool shall not adjust the GivenNameSegmentScore (i.e., DELETE TAQs None Occur, DISREGARD TAQs >=1 Match).
        - If DELETE TAQs are present in both comparands and no matches are found, then the GivenNameSegmentScore will be multiplied by the GivenNameTAQDeleteFactor (i.e., DELETE TAQs No Match, DISREGARD TAQs >=1 Match).
        - If DELETE TAQs are present in both comparands and a match is found, then the tool shall not adjust the GivenNameSegmentScore (i.e., DELETE TAQs >=1 Match, DISREGARD TAQs >=1 Match).
        - If DELETE TAQs are present in one comparand, but not the other, then the GivenNameSegmentScore will be multiplied by the GivenNameTAQDeleteAbsentFactor (i.e., DELETE TAQs Absent, DISREGARD TAQs >=1 Match).
- If no match is found, then the GivenNameSegmentScore will be multiplied by the GivenNameTAQDisregardFactor (i.e., DELETE TAQs >=1 Match, No Match, Absent, None Occur, DISREGARD TAQs No Match).
- If both comparands have all DELETE TAQs associated with them, the tool shall determine if any of the DELETE TAQs match:

- If there is any match, then the tool shall not adjust the GivenNameSegmentScore (i.e., DELETE TAQs >=1 Match, DISREGARD TAQs None Occur).

- If there is no match, then the GIvenNameSegmentScore will be multiplied by the GivenNameTAQDeleteFactor (i.e., DELETE TAQs No Match, DISREGARD TAQs None Occur).

The following table describes the conditions governing the application of TAQ as described in the text above:

DELETE TAQ(s)	DISREGARD TAQ(s)	Impact on GivenNameSegmentScore
None Occur	None Occur	No Change
None Occur	No Match	Apply GivenNameTAQDisregardFactor
None Occur	Absent	Apply GivenNameTAQDisregardAbsentFactor
None Occur	>=1 Match	No Change
No Match	None Occur	Apply GivenNameTAQDeleteFactor
No Match	No Match	Apply GivenNameTAQDisregardFactor
No Match	Absent	Apply GivenNameTAQDisregardAbsentFactor
No Match	>=1 Match	Apply GivenNameTAQDeleteFactor
Absent	None Occur	Apply GivenNameTAQDeleteAbsentFactor
Absent	No Match	Apply GivenNameTAQDisregardFactor
Absent	Absent	Apply GivenNameTAQDisregardAbsentFactor
Absent	>=1 Match	Apply GivenNameTAQDeleteAbsentFactor
>=1 Match	None Occur	No Change
>=1 Match	No Match	Apply GivenNameTAQDisregardFactor
>=1 Match	Absent	Apply GivenNameTAQDisregardAbsentFactor
>=1 Match	>= 1 Match	No Change

"Match" in the table indicates that the stated TAQ Type occurs in both comparands and at least one of the TAQ Type values occurs in both comparands, i.e., the TAQ Type values are the same. For example, the DELETE TAQ value "Mr" may occur in both comparands.

"No Match" in the table indicates that the stated TAQ Type occurs in both comparands but none of the TAQ Type values occurs in both comparands, i.e., the values are not the same. For example, the single DELETE TAQ value "Mr" may occur in one comparand and the single DELETE TAQ value "Mrs" may occur in the other comparand.

"Absent" in the table indicates that the stated TAQ Type is absent in one of the comparands but occurs in the other comparand. For example, the DELETE TAQ value "Mr" may occur in one comparand and there may be no DELETE TAQ value at all in the other comparand.

"None Occur" in the table indicates that the stated TAQ Type does not occur in either comparand. For example, no DELETE TAQs occur in either comparand.

6.1.2.2.4.2 Future Version Notes

- The GivenNameTAQDisregardFactor will be replaced by the highest TAQ-DISREGARD-WEIGHT for each DISREGARD TAQ relationship that is defined in a TAQ-DISREGARD-WEIGHT Table. The TAQ-DISREGARD-WEIGHT Table defines a weighted relationship between two TAQs that occur within a specified cultural boundary or partition. There may be a default TAQ-DISREGARD-WEIGHT established so that only those TAQ relationships that warrant special weighting may be entered into the TAQ-DISREGARD-WEIGHT Table.

6.1.2.3 Determine GivenNameScore

In order to determine the GivenNameScore, the tool shall:

- **Compute the Highest GivenNameSegmentScore(s) (SurnameMode="Highest");**
- **Compute the Best Combination of GivenNameSegmentScore(s) (SurnameMode="Average"); and**
- **Compute the Lowest GivenNameSegmentScore(s) (SurnameMode="Lowest");**

If either comparand has just one GN segment, then the tool shall set the GivenNameScore = the Highest (Best) GivenNameSegmentScore found in the evaluation matrix.

If more than one segment occurs in both Given Names, then the tool shall **Apply the Given Name Mode** in its determination of the GivenNameScore.

6.1.2.3.1 Compute Highest GivenNameSegmentScore(s) (GivenNameMode="Highest")

The tool shall compute the highest set of GivenNameSegmentScores (includes the highest GivenNameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that includes the highest set of GivenNameSegmentScores.

In the following example, the highest GivenNameSegmentScores will be 1.0 and .57, since 1.0 is the highest GivenNameSegmentScore.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

In the following example, the highest GivenNameSegmentScores will be 1.0 and .62, since 1.0 is the highest GivenNameSegmentScore, and .62 is the next highest GivenNameSegmentScore that is in a different row and column combination in the matrix.

	Garcia	Garza	Garza
Garica	.57	.62	.62
Garza	.62	1.0	1.0

6.1.2.3.2 Compute Best Combination of GivenNameSegmentScore(s) (GivenNameMode="Average")

The tool shall compute the best possible combination of scores (i.e., the Highest Average of GivenNameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that gives the highest sum.

In the following example, the best combination of GivenNameSegmentScores will be 1.0 and .57, since ((1.0+.57)/2) > ((.62+.62)/2) = .79 > .62.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

6.1.2.3.3 Compute Lowest GivenNameSegmentScore(s) (GivenNameMode="Lowest")

The tool shall compute the lowest set of GivenNameSegmentScores (includes the Lowest GivenNameSegmentScores) from the evaluation matrix of scores.

During the evaluation of the matrix, a given row or column shall contribute one and only one score.

The tool shall select the combination of matrix values (with no row or column being used more than once) that includes the lowest set of GivenNameSegmentScores.

In the following example, the lowest GivenNameSegmentScores will be .57 and 1.0, since .57 is the lowest GivenNameSegmentScore.

	Garcia	Garza
Garica	.57	.62
Garza	.62	1.0

In the following example, the lowest GivenNameSegmentScores will be .57 and 1.0, since .57 is the lowest GivenNameSegmentScore, and 1.0 is the next lowest GivenNameSegmentScore that is in a different row and column combination in the matrix.

	Garcia	Garza	Garza
Garica	.57	.62	.62
Garza	.62	1.0	1.0

6.1.2.3.4 Apply Given Name Mode (GivenNameMode)

If GivenNameMode = "highest", the tool shall set the GivenNameScore = the highest GivenNameSegmentScore found in the evaluation matrix.

If GivenNameMode = "average", the tool shall set the GivenNameScore = the average of the GivenNameSegmentScores found in the evaluation matrix.

If GivenNameMode = "lowest", the tool shall set the GivenNameScore = the lowest GivenNameSegmentScore found in the evaluation matrix.

6.1.2.3.5 Determine GivenNameCompressedScore (GivenNameCheckCompressed, GivenNameCompressedScore)

This function handles names that are essentially the same name but are segmented differently (e.g., "Anne Marie" → "AnneMarie").

If GivenNameCheckCompressed = "T", then the tool shall generate a GivenNameCompressedScore in the following manner:

- Create query and candidate Compressed GN fields from their original GN fields, by processing segmentation and removal markers, and then eliminating all remaining blanks.
- Compare the query and candidate COMPRESSED GN fields to determine if there is an exact match.
- If there is an exact match of the COMPRESSED GN fields, the tool shall set the GivenNameScore to the higher score of the GivenNameCompressedScore or the previously calculated GivenNameScore.

6.1.3 Determine if SurnameScore exceeds SurnameThreshold

The tool shall determine whether the candidate name is a potential match by checking to see if the SurnameScore exceeds the SurnameThreshold prior to returning the candidate name as a potential match.

If the SurnameScore exceeds the SurnameThreshold, then the tool shall determine that the candidate name is still a potential match.

If the SurnameScore does not exceed the SurnameThreshold, then the tool shall determine that the candidate name is no longer a potential match.

Even if the candidate name is no longer considered a potential match, the tool shall continue processing in the event that all evaluated names were requested to be scored and returned marked as match or no match.

6.1.4 Determine if GiveNameScore exceeds GivenNameThreshold

The tool shall determine whether the candidate name is a potential match by checking to see if the GivenNameScore exceeds the GivenNameThreshold prior to returning the candidate name as a potential match.

If the GivenNameScore exceeds the GivenNameThreshold, then the tool shall determine that the candidate name is still a potential match.

If the GivenNameScore does not exceed the GivenNameThreshold, then the tool shall determine that the candidate name is no longer a potential match.

Even if the candidate name is no longer considered a potential match, the tool shall continue processing in the event that all evaluated names were requested to be scored and returned marked as match or no match.

6.1.5 Compute NameScore & Determine If Potential Match (NameThreshold, SurnameWeight, GivenNameWeight)

If the SurnameWeight = GivenNameWeight = 0, then the tool shall set NameScore = 0.

If the SurnameWeight = GivenNameWeight <> 0, then the tool shall assign NameScore = (SurnameScore + GivenNameScore)/2.

If the SurnameWeight <> GivenNameWeight then the tool shall assign

$$\text{NameScore} = \frac{(\text{SurnameScore*SurnameWeight}) + (\text{GivenNameScore*GivenNameWeight})}{(\text{SurnameWeight} + \text{GivenNameWeight})}$$

The tool shall then determine whether the candidate name is a potential match by checking to see if the NameScore exceeds the NameThreshold prior to returning the candidate name as a potential match.

If the NameScore exceeds the NameThreshold, then the tool shall determine that the candidate name is still a potential match.

Developers will be able to establish their own method to determine if a candidate name is a potential match. This will enable developers to integrate other data elements and other criteria in the final name score, if desired. Note that developers may or may not choose to utilize the SurnameWeight and GivenNameWeight factors in their method.

The tool shall populate the Results List with a candidate name based on whether it is identified as a potential match. If no Results List is being constructed, then the tool shall return the candidate name and its associated scores.

6.2 Design Notes

1. We considered performing an exact match on the name prior to performing the "fuzzy" matching, but decided that the overhead of checking every candidate name as an exact match was more than the cost of performing the "fuzzy" match when there is an exact match – our assumption is that the tool will be evaluating more similar names than exact match names.

2. The following parameters have been supported by earlier versions of DNC and are not proposed to be included in the tool.

    - The following parameters were used with the DP2-based pass-1 search for DNC, and were supplanted by the COF processor and, therefore, are now no longer functional in DNC:

        - **KICKOUT**
        - **PARTITION**
        - **TEST VALUE**
        - **PROXRETURN**

    - The following parameters were specifically supported in DNC for the State Department:

        - **REFULEVx (REFULEV0 - REFULEV4)**
        - **DOBFACTOR**
        - **FIXLASTSEG**
        - **CHKSUBSTR – (DNC does not use except for 1 COB)**
        - **SUBSCORE – (DNC does not use)**
        - **MINSEGLEN – (DNC does not use)**
        - **LTRIGRAPH - (related to COF processing)**
        - **NTRIGRAPH - (related to COF processing)**

7. Produce and Manage Results

7.1 Functionality

7.1.1 Define Criteria for Results List

7.1.1.1 Functionality

The Results list is defined as either an unordered candidate name list, or an ordered candidate name list ordered by the relative probability that each candidate name is similar to a specified query name. This probability is represented by the final NameScore. The Results List may include both similar and dissimilar names, depending on the criteria for defining the results.

When a set of candidate names is to be evaluated, the tool shall enable the developer to define the criteria for producing and managing their own Results List. These criteria shall include:

- establishing a type of Results List :
    - 1 = an unordered list of all candidate names whose name score exceeds a pre-defined name threshold (e.g., if the threshold = 0, all candidate names will be returned in an unordered list);
    - 2 = an ordered list of all candidate names whose name score exceeds a pre-defined name threshold (e.g., if the threshold = 0, all candidate names will be returned in an ordered list);
    - 3 = an ordered list of the top X candidate names whose name score exceeds a pre-defined name threshold, and where X is a number;
- establishing a size limit for the Results list, which in effect defines the value of X for producing the top X candidate names;
- defining a SurnameThreshold;
- defining a GivenNameThreshold; and
- defining a NameThreshold.

If the NameThreshold is set to 0, the tool shall return all candidate names in a Results List, unless a Results List size limit is established.

7.1.1.2 Design Notes

1. The developer can define the top X candidate names by establishing their own Results list and attaching it to the query name. In establishing their own Results list, the developer is in effect specifying its size, i.e., the value of X.

7.1.2 Produce Results

Only those candidate names whose NameScore is greater than the NameThreshold will be included in the Result List.

The tool shall produce the Result List by sorting the candidate names in descending order by their NameScore.

If two candidate NameScores are the same, then the tool shall order the same scored candidate data according to the following rules:

- first order the candidate names in descending order by each candidate's SurnameScore.
- if two candidate's SurnameScores are the same, then do the following:
    - if SurnameMode = "average" or "lowest", then do the following:
        - first order the candidate names in descending order by each candidate's GivenNameScore.
        - if two candidate's GivenNameScores are the same, then do the following:
            - If GivenNameMode = "average" or "lowest", then do the following:
                - first order the candidate names in ascending order by the difference in the number of SN segments between the candidate name and the query name.
                - if the difference in the number of SN segments between the candidate name and the query name is the same, then order the candidate names in ascending order by the difference in the number of GN segments between the candidate name and the query name.
            - If GivenNameMode = "highest", then do the following:
                - first order the candidate names in descending order by each candidate's next highest GivenNameSegmentScore.
                - if two candidate's next highest GivenNameSegmentScores are still all the same, then continue to evaluate the next highest GivenNameSegmentScores (up to n, where n is the greater

number of Given Name Segments in the two evaluation Given Names), and order the candidate names in descending order by each candidate's next highest GivenNameSegmentScore. If one of the evaluation Given Names has fewer segments than the other evaluation Given Name, its missing GivenNameSegmentScores will be set to .50 in order to evaluate them.

- if two candidate's GivenNameSegmentScores are all the same, then order the candidate names in ascending order by the difference in the number of SN segments between the candidate name and the query name.
- if the difference in the number of SN segments between the candidate name and the query name is the same, then order the candidate names in ascending order by the difference in the number of GN segments between the candidate name and the query name.

- If SurnameMode = "highest", then do the following:
    - first order the candidate names in descending order by each candidate's next highest SurnameSegmentScore..
    - if two candidate's next highest SurnameSegmentScores are still all the same, then continue to evaluate the next highest SurnameSegmentScores (up to n, where n is the greater number of Surname Segments in the two evaluation surnames), and order the candidate names in descending order by each candidate's next highest SurnameSegmentScore. If one of the evaluation surnames has fewer segments than the other evaluation surname, its missing SurnameSegmentScores will be set to .50 in order to evaluate them.
        - if two candidate's next highest SurnameSegmentScores are all the same, then do the following:
            - first order the candidate names in descending order by each candidate's GivenNameScore.
            - if two candidate's GivenNameScores are the same, then do the following:
                - if GivenNameMode = "average" or "lowest", then do the following:

        - first order the candidate names in ascending order by the difference in the number of SN segments between the candidate name and the query name.

        - if the difference in the number of SN segments between the candidate name and the query name is the same, then order the candidate names in ascending order by the difference in the number of GN segments between the candidate name and the query name.

    - if GivenNameMode = "highest", then do the following:

        - first order the candidate names in descending order by each candidate's next highest GivenNameSegmentScore.

        - if two candidate's next highest GivenNameSegmentScores are still all the same, then continue to evaluate the next highest GivenNameSegmentScores (up to n, where n is the greater number of Given Name Segments in the two evaluation Given Names), and order the candidate names in descending order by each candidate's next highest GivenNameSegmentScore. If one of the evaluation Given Names has fewer segments than the other evaluation Given Name, its missing GivenNameSegmentScores will be set to .50 in order to evaluate them.

            - if two candidate's GivenNameSegmentScores are all the same, then order the candidate names in ascending order by the difference in the number of SN segments between the candidate name and the query name.

            - if the difference in the number of SN segments between the candidate name and the query name is the same, then order the candidate names in ascending order by the difference in the number of GN segments between the candidate name and the query name.

If the developer specified the top X option, then the tool shall produce a Result List that contains the X candidate names that are determined after sorting, to be the most likely matches.

The tool shall provide the developer the capability to establish custom results ordering methods, if desired.

7.1.3 Retrieve Results

The tool shall provide the developer the capability to retrieve matched candidate names from the Results List and retrieve additional information about the candidate names to include at a minimum, the Given Name field, the Surname field, the GivenNameScore, the SurnameScore, and other data that the developer may have defined.

8. EVALUATION FACTORS and PARAMETERS

8.1 SurnameCheckInitial (previously known as ISSNINITL)

The SurnameCheckInitial indicates whether a single character in the surname segment will be treated as an initial. When SurnameCheckInitial = "T", single characters in the query SN segment are treated as initials and, if they match on a candidate SN segment, the tool will usually set the SurnameSegmentScore = SurnameInitialScore (except when there is an exact match on initials, in which case the SurnameSegmentScore = (1+SurnameInitialScore)/2 : Exact matches on initials are not considered "exact matches" because the initial may represent two different name segments).

Initials are relatively uncommon in the surname field. In some cases, such as Chinese names, single characters are common in the surname field, in which case one will not want a single letter to be treated as an initial, but rather, treated as a name. If treated as a name, a single character will be analyzed using digraph matching, and will generally be given very little value, unless it matches on an identical one character name. In such cases, SurnameCheckInitial should be set to "F".

SurnameCheckInitial
Possible Settings: {T,F}
Default: {F}

8.2 SurnameCheckVariant (previously known as CHKVARIANT)

In many cases there are variant spellings for a surname that do not share many common digraphs. One possible solution to this problem is the use of the SurnameCheckVariant parameter. When SurnameCheckVariant = "T", a table containing Surname Variants is referenced during the evaluation as well.

SurnameCheckVariant
Possible Settings: {T, F}
Default: {F}

8.3 SurnameCheckBias (previously known as LDIBIAS)

The SurnameCheckBias was designed to aid in the analysis of Russian names and other naming systems which make use of complex morphological endings. If SurnameCheckBias = "T", the tool places more emphasis upon the beginning digraphs of a particular name and de-emphasizes later digraphs. This is done to eliminate the effect of the morphological endings common to Russian names. Because the names are generally long, and many of the names end with the same endings (such as -ovich), candidates that were not very good were being easily returned because there were many digraph matches. When SurnameCheckBias = "T", and when calculating a SurnameSegmentScore, the first digraph in the both the query and candidate SN segments is assigned a weight factor of 1.00, the second digraph is assigned a weight factor of.9, and so forth until .1 is reached, at which point, the remainder of the digraphs are assigned a weight factor of .1.

If GivenNameCheckBias = "T", the tool shall apply bias by:

- First, assigning the first contributing digraph in each GN segment a weight factor of 1.00, the second contributing digraph a weight factor of .9, and so on until the tenth contributing digraph is reached, at which point, all remaining digraphs shall be assigned a weight factor of .1.
- then determine which digraphs match between the two GN segments;
- sum the weight factors assigned to the matched query GN digraphs with the weight factors assigned to the matched candidate GN digraphs; and
- divide by the sum of all contributing digraphs for both the query GN and the candidate GN.

GivenNameCheckBias : T

Query : Moskyovich, FNU
Candidate : Markovich, FNU

-M Mo os sk ky yo ov vi ic ch h-
(1.0) (.9) (.8)(.7)(.6) (.5) (.4)(.3)(.2) (.1)(.1)

-M Ma ar rk ko ov vi ic ch h-
(1.0) (.9) (.8)(.7)(.6) (.5)(.4)(.3)(.2) (.1)

The following digraphs match:

	-M	ov	vi	ic	ch	h-
Query Weight Factors:	(1.0)	(.4)	(.3)	(.2)	(.1)	(.1)
Candidate Weight Factors:	(1.0)	(.5)	(.4)	(.3)	(.2)	(.1)

$$\frac{\text{Matched Digraphs}}{\text{Total possible Digraphs}} = \frac{(1.0+.4+.3+.2+.1+.1)+(1.0+.5+.4+.3+.2+.1)}{(1.0+.9+.8+.7+.6+.5+.4+.3+.2+.1+.1)+(1.0+.9+.8+.7+.6+.5+.4+.3+.2+.1)}$$

Therefore, the GivenNameSegmentScore == (4.6 / 11.1) = 0.41

There are some problems that may occur when SurnameCheckBias = "T." For example, SurnameCheckBias was set to "T" during the LQA for Poland because of the high frequency morphological endings. However, as a result, a common surname root such as "Kowal" returned names such as "Kowalczyk," "Kowalewska," "Kowalewski," "Kowalska, "Kowalik," "Kowalow," "Kowal," and "Kowalkowska."(Memo# L94289) Therefore, names which were often not good hits were returned because of the additional weight placed upon the beginning digraphs in a name. Please see memos L94289 and L94290 for further explanation of the possible problems associated with SurnameCheckBias.

When the SurnameCheckBias = "T", more emphasis is placed upon the beginning digraphs of a name. When the SurnameCheckBias = "F", equal value is placed upon all matching digraphs.

SurnameCheckBias
Possible Settings: {T,F}
Default: {F}

8.4 SurnameCheckUnknownNotExist, LastNameUnknownScore, NoLastNameScore

Surname Check Non-existent value – used to assign a higher score to a SN segment if there is no value in one comparand SN segment, yet there is a value in the other comparand.

Query : Malcolm LNU
Candidate : Malcolm Shabaz

If SurnameCheckUnknownNotExist = "F", the SurnameSegmentScore for "LNU" compared to "Shabaz" would be 0. With SurnameCheckUnknownNotExist = "T", the SurnameSegmentScore for "LNU" compared to "Shabaz" will be set to the LastNameUnknownScore. The LastNameUnknownScore will be set fairly high to accommodate for missing values when it is unclear whether there should or should not be a value. This would result in a higher SurnameSegmentScore which means that the candidate will be more likely to appear in the TOP X results as well as exceed the NameThreshold if it is set above 0.

SurnameCheckUnknownNotExist : T

Query : Malcolm NLN

Candidate : Malcolm Shabaz

In the second example above, the SurnameSegmentScore for "NLN" compared to "Shabaz" will be set to the NoLastNameScore. The NoLastNameScore will typically be very low to accommodate for the fact that there is no last name defined in the query but a last name appears in the candidate. Thus, the candidate is not a likely match.

SurnameCheckUnknownNotExist
Possible Settings: {T, F}
Default: {F}

NoLastNameScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.80}

LastNameUnknownScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.85}

8.5 SurnameCheckCompressed, SurnameCompressedScore

SurnameCheckCompressed
Possible Settings: {T,F}
Default: {F}

SurnameCompressedScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.9}

8.6 SurnameAnchorSegment, SurnameAnchorFactor (previously known as ANCHSEG, ANCHVAL)

In order to determine the relative position in both comparands, the tool shall establish an "index" of a segment based on the SurnameAnchorSegment. For SurnameAnchorSegment = "none" or "first", the tool shall count segments from left to right. For SurnameAnchorSegment = "last", the tool shall count segments from right to left.

Either SurnameOutOfPositionFactor or SurnameAnchorFactor, but not both, can be applied to the SurnameSegmentScore.. Thus, if SurnameOutOfPositionFactor has already been applied, then the SurnameAnchorFactor can not be applied, even if the SurnameAnchorSegment = "first" or "last". If SurnameOutOfPositionFactor does not apply, then the SurnameAnchorFactor may apply if SurnameMode is "highest" or "lowest". If SurnameMode is "average", the SurnameAnchorFactor is not applied.

The SurnameAnchorSegment is used in compound surnames to emphasize the importance of one segment of the name over another. For example, when dealing with Portuguese names, it is the second (i.e., last) surname which is more important, whereas when dealing with other Hispanic names it is generally the first surname which is of primary importance.

When the SurnameAnchorSegment = "none", neither segment in the name is given more weight (i.e., basically the SurnameAnchorSegment is turned off). When the SurnameAnchorSegment = "first", both comparand segments are left-aligned and the left-most or first segment in the name is given more weight. When the SurnameAnchorSegment = "last", both comparand segments are right-aligned and the last segment in the name is given more weight. Thus, if only two surname segments exist, then the right-most or last segment is given more weight if SurnameAnchorSegment = "last".

The way in which the SurnameAnchorSegment gives more weight to certain name segments is through the use of the SurnameAnchorFactor. For example, if the SurnameAnchorSegment = "first", but neither of the similar digraph names are in the first position, then the SurnameSegmentScore is multiplied by the SurnameAnchorFactor.

SurnameAnchorSegment : first
SurnameOutOfPositionFactor : .65
SurnameMode : highest

SurnameAnchorFactor : .70

Query : Lopez Garcia, Luis
Candidate : Santos Garcia, Luis

In this example, since the SurnameAnchorSegment = first, the two comparands are left-aligned. After left-aligning the comparands, more weight should be given to the name in the first position. The name "Garcia" matches; however it is in the last position in both the query and the candidate (i.e., it is not in the first position, which is the SurnameAnchorSegment). Therefore, the SurnameSegmentScore(1.00) is multiplied by the SurnameAnchorFactor(.70), thus yielding a score of : SurnameSegmentScore * SurnameAnchorFactor = 1.00 * .70 = .70. Because the SurnameMode = "highest", the SurnameScore = the highest SurnameSegmentScore (1.0). Therefore, the way in which one surname is given more weight is actually to devalue the other or give it less weight.

If the same parameter settings are in effect, and the segments in the first position in both comparands are being compared, then the SurnameSegmentScore is not devalued. For example:

SurnameAnchorSegment : first
SurnameOutOfPositionFactor : .60
SurnameMode : highest

SurnameAnchorFactor : .70

Query : Gonzalez Garcia, Mario
Candidate : Gonzalez Salvador, Mario

In this case, after left-alignment, the name "Gonzalez" is considered to be in the first position in both comparands, and since the SurnameAnchorSegment ="first", it receives a SurnameSegmentScore of 1.00.

Similarly, if the SurnameAnchorSegment = "last", the emphasis is upon the last element in the Surname. In the following example, after right-alignment, the names "Lopez" and "Santos" do not share any common digraphs, and therefore receive a SurnameSegmentScore of 0. However, the name "Garcia" matches and is in the second position in comparands, therefore it receives a score of 1.00 and is not multiplied by the SurnameAnchorFactor since the SurnameAnchorSegment = "last". Since the SurnameMode = "highest", the SurnameScore = 1.00, which is the highest SurnameSegmentScore.

SurnameAnchorSegment : last
SurnameOutOfPositionFactor : .65
SurnameMode : highest
SurnameAnchorFactor : .70

Query : Lopez Garcia, Luis
Candidate : Santos Garcia, Luis

In contrast, if the evaluated segments, (i.e., Lopez and Santos in the example above) are not in the last position, the SurnameSegmentScore will be multiplied by the SurnameAnchorFactor.

SurnameAnchorSegment : last

Query : Gomez Hernandez, Mario
Candidate : Gomez Lopes, Mario

Although the name "Gomez" is an exact match, it is not in the last position in either comparand, and the SurnameAnchorSegment = "last". Therefore, "Gomez" is multiplied by the SurnameAnchorFactor. The SurnameSegmentScore(1.00) is multiplied by the SurnameAnchorFactor (.70) producing the SurnameSegmentScore = (.70).

The value of the SurnameAnchorSegment determines which of the name segments, if any, is to receive the most weight. Raising the SurnameAnchorFactor will actually give more value to the segment that is not the SurnameAnchorSegment, while lowering the SurnameAnchorFactor will lower the value of the segment that is not the SurnameAnchorSegment.

SurnameAnchorSegment
Possible Settings: {first, last, none}
Average Range: {first, last, none}
Default: {none}

SurnameAnchorFactor
Possible Settings: {0.00, 0.01,... 1.00}
Average Settings: {.50...70}
Default: {.70}

8.7 SurnameCheckTAQ

When the SurnameCheckTAQ = "off", no TAQ processing will take place at all.

When the SurnameCheckTAQ = "remove", then TAQ(s) will simply be removed from the name data.

When the SurnameCheckTAQ = "score", TAQ(s) will be identified, removed, and associated with each relevant name segment during preprocessing, and then the SurnameTAQDeleteFactor, SurnameTAQDeleteAbsentFactor, SurnameTAQDisregardFactor, and SurnameTAQDisregardAbsentFactor, will be multiplied against the SurnameSegmentScore, which will in effect reduce the value of the SurnameSegmentScore.

SurnameCheckTAQ
Possible Settings: {off, remove, score}
Default: {score}

8.8 SurnameMode (previously known as SNMODE)

The SurnameMode can be set to "highest", "average", or "lowest" depending upon how flexible or stringent one wants the parameters to be. "Highest" is the most flexible mode setting and "lowest" is the most stringent setting. SurnameMode only has an effect if there is more than one name in the surname. If there is more than one name in the surname, and the SurnameMode = "highest", then the SurnameScore will be set to the highest SurnameSegmentScore.

SurnameMode : highest

Query : Lopez Garcia, Maria
Candidate: Lopez Gonzalez, Maria

In this example, the highest SurnameSegmentScore will be used to evaluate the surname :"Lopez

Gonzalez". "Lopez" in the candidate matches "Lopez" in the query exactly, thus receiving a score of 1.00. Since "Gonzalez" has very few digraph matches with "Garcia", the SurnameScore will be set to 1.00, which is the highest SurnameSegmentScore.

If the SurnameMode = "average", the SurnameScore will be set to the average of the SurnameSegmentScores.

SurnameMode : average

Query : Lopez Garcia, Maria
Candidate: Lopez Gonzalez, Maria

The candidate segment "Lopez" matches the query segment "Lopez" with a score of 1.00. However, there is only one digraph match between "Garcia" and "Gonzalez", yielding a score of 1/9 or .11. Since SurnameMode = "average", the SurnameScore will be set to the average of the two SurnameSegmentScores. The average of the two SurnameSegmentScores in this example is (1+.11)/2 = .56.

If the SurnameMode = "lowest", and if there is more than one name in the surname, then the SurnameScore will be set to the lowest SurnameSegmentScore. In the example illustrated above, the SurnameScore will be set to .11. Clearly, SurnameMode = "lowest" is the most stringent setting.

If a threshold is defined to identify "hits", then setting the SurnameMode to "highest" will result in the return of more hits. Raising the SurnameMode to "average" will decrease the number of hits returned since the average score of the surnames must also pass the threshold. Raising the SurnameMode to "lowest" will further decrease the number of hits returned since both names must pass the threshold.

SurnameMode:
Possible Settings: {highest, average, lowest}
Average Range: {highest, average, lowest}
Default: {average}

8.9 SurnameExactInitialMatchScore

If SurnameCheckInitial is set to True, then the SurnameExactInitialMatchScore is used to indicate whether two single characters that match one another should be considered "exact matches", and therefore be assigned a score of 1.0. In some cases, it may be desirable to not consider two single characters as an exact match since it is possible that the two characters may represent two different names. In these cases, one might want to set the SurnameExactInitialMatchScore = (1-SurnameInitialScore)/2.

SurnameExactInitialMatchScore
Possible Settings: {0.00, 0.1, ... 1.00}

Average Settings: {1.0}
Default: {1.0}

8.10 SurnameInitialScore

The SurnameInitialScore behaves in the same manner as the GivenNameInitialScore but it applies to surnames rather than given names. This parameter will be useful in dealing with Hispanic surnames which frequently use an initial to represent High Frequency second surnames.

SurnameInitialScore
Possible Settings: {0.00, 0.1, ... 1.00}
Average Settings: {.60....90}
Default: {.85}

8.11 SNV-SCORE

The SNV-SCORE is the value given to a pair of Surname variants found in the SURNAME-VARIANT Table. The SNV-SCORE is generally set very high, usually at .95.

SNV-SCORE
Possible Settings: {0.00, 0.01, ... 1.00}
Default: {defined by variant pair}

8.12 SurnameOutOfPositionFactor, SurnameAnchorSegment (previously known as SNOOPS, ANCHSEG)

In order to determine the relative position of name segments in both the query and candidate, the tool shall establish an "index" of a segment based on the SurnameAnchorSegment. For SurnameAnchorSegment = "none" or "first", the tool shall left-align the name segments. For SurnameAnchorSegment = "last", the tool shall right-align the name segments.

The SurnameOutOfPositionFactor factor only applies to name segments that are out of position (i.e., not in the same relative position). When a surname segment is out of position, the SurnameSegmentScore is multiplied by the SurnameOutOfPositionFactor factor. In the following example, after left-aligning, the candidate name segments "Garcia" and "Gonzalez" are both considered to be out of position.

SurnameMode : average
SurnameOutOfPositionFactor : .65
SurnameAnchorSegment : first

Query : Gacria Gonzalez, Mario
Candidate : Gonzalez Garcia, Mario

The name "Gonzalez" is an exact match but it is out of position. Therefore, it receives a value of .65 (SurnameOutOfPositionFactor) X 1.00 (SurnameSegmentScore) = .65.

The name "Garcia" is not an exact match. Of the fourteen digraphs, there are 4 digraph matches.

The total possible digraphs are:

#G Ga ar rc ci ia a#
#G Ga ac cr ri ia a#

The matched digraphs are:

#G Ga ia a#
#G Ga ia a#

Therefore, the name receives a score of .65 (SurnameOutOfPositionFactor) X .57 (SurnameSegmentScore = 8/14 = .57).

The SurnameMode in this example = "average". Therefore, the SurnameScore = the average of the two SurnameSegmentScores, which is.51 = ((.65+.51)/2).

8.12.1 SurnameOutOfPositionFactor With Surnames Containing Only 1 Name Segment

In the following example, the name "Sanchez" is considered to be out of position.

SurnameAnchorSegment : first

Query : Ramirez Sanchez, Luis
Candidate : Sanchez, Luis

In the query "Sanchez" is considered to be in the last position, whereas, in the candidate, "Sanchez' is considered to be in the first position. Therefore, the SurnameScore = SurnameOutOfPositionFactor multiplied by the SurnameSegmentScore (1.00).

If the SurnameAnchorSegment = "last", then Sanchez would be considered to be in position, and the SurnameOutOfPositionFactor would not be applied.

If a NameThreshold is defined, raising the SurnameOutOfPositionFactor will generally result in the return of more names and lowering the SurnameOutOfPositionFactor will make it more difficult for names to pass the threshold.

SurnameOutOfPositionFactor:
Possible Settings: {0.00, 0.01,... 1.00}
Average Range: {.50...70}
Default: {.60}

8.13 SurnameTAQDisregardAbsentFactor

absent Surname Disregard TAQ score – refer to section on TAQ scoring in main document for description.

SurnameTAQDisregardAbsentFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.80}

8.14 SurnameTAQDeleteAbsentFactor

absent Surname Delete TAQ score – refer to section on TAQ scoring in main document for description.

SurnameTAQDeleteAbsentFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.90}

8.15 SurnameTAQDeleteFactor

delete Surname TAQ score – refer to section on TAQ scoring in main document for description.

SurnameTAQDeleteFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.85}

8.16 SurnameTAQDisregardFactor

disregard Surname TAQ score – refer to section on TAQ scoring in main document for description.

SurnameTAQDisregardFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.7}

8.17 LastNameUnknownScore

If one of the comparands has been identified as having "last name unknown", then the segment score assigned when comparing that comparand with another is the LastNameUnknownScore.

LastNameUnknownScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.6}

8.18 NoLastNameScore

If one of the comparands has been identified as having "no last name", then the segment score assigned when comparing that comparand with another is the NoLastNameScore.

NoLastNameScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.65}

8.19 SurnameCompressedScore

In some instances, TAQ values become conjoined with stems in unpredictable ways. In some instances, two surname comparands are exact matches except for spacing (e.g., "de la Garcia" and "delaGarcia"). If this is determined to be the case, the tool will assign the SurnameCompressedScore to the SurnameScore.

SurnameCompressedScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.9}

8.20 SurnameThreshold (previously known as SNTHRESH)

The SurnameThreshold is the threshold which the SurnameScore must exceed in order for the candidate name to be included in the Results list. If a developer wants to define a threshold rather than return the TOP X names, then this parameter may be set to some value other than 0. Setting the SurnameThreshold to 0 essentially turns off the SurnameThreshold. As the SurnameThreshold is raised, fewer candidate names will be returned as it will be more difficult for a candidate name to pass the higher SurnameThreshold. Conversely, as the SurnameThreshold is lowered, more candidate names will be returned as it will be easier for a candidate name to pass the lower SurnameThreshold.

SurnameThreshold
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.50}

8.21 SurnameWeight

The SurnameWeight is the factor (weight) that can be applied to the SurnameScore when determining whether a candidate name is to be included in the Results list. This weight factor enables one to assign more or less emphasis to a potential candidate based on the SurnameScore.

The higher the SurnameWeight, the greater the value of the SurnameScore contribution to the overall NameScore. If the SurnameWeight is set to 0, the SurnameScore will not contribute any value to the overall NameScore. In Version 1, the exception to this occurs if the GivenNameWeight is also set to 0, in which case, the weight factors cancel one another out. In Version 1, we multiply the SurnameScore by the SurnameWeight as part of the default overall NameScore calculation. Note that developers may or may not choose to apply the SurnameWeight when calculating an overall NameScore if they create a different scoring algorithm.

SurnameWeight
Possible Settings: {0.0, 0.1, ...1.0}
Default: {1.0}

8.22 GivenNameCheckInitial (previously known as ISGNINITL)

The GivenNameCheckInitial behaves the same as SurnameCheckInitial, but applies to given names rather than surnames.

GivenNameCheckInitial
Possible Settings: {T, F}
Default: {T}

8.23 GivenNameCheckVariant (previously known as CHKVARIANT)

The GivenNameCheckVariant behaves in the same manner as the SurnameCheckVariant, but applies to given names rather than surnames. When SurnameCheckVariant = "T", a table containing GN Variants is referenced during the evaluation as well.

GivenNameCheckVariant
Possible Settings: {T, F}
Default: {T}

8.24 GivenNameCheckBias

This parameter behaves in the same manner as the SurnameCheckBias but it applies to given names rather than surnames.

GivenNameCheckBias
Possible Settings: {T,F}
Default: {F}

8.25 GivenNameCheckUnknownNotExist, NoFirstNameScore, FirstNameUnknownScore

GivenNameCheckUnknownNotExist is similar to SurnameCheckUnknownNotExist except that it applies to the GN field. The parameters for GivenNameCheckUnknownNotExist are also specific to the GN field.

GivenNameCheckUnknownNotExist
Possible Settings: {T, F}
Default: {F}

NoFirstNameScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.80}

FirstNameUnknownScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.85}

8.26 GivenNameCheckCompressed, GivenNameCompressedScore

GivenNameCheckCompressed
Possible Settings: {T,F}
Default: {F}

GivenNameCompressedScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.9}

8.27 GivenNameAnchorSegment, GivenNameAnchorFactor

The GivenNameAnchorSegment behaves in the same manner as the SurnameAnchorSegment but it applies to given names rather than surnames.

The GivenNameAnchorFactor behaves in the same manner as the SurnameAnchorFactor but it applies to given names rather than surnames.

GivenNameAnchorSegment
Possible Settings: {first, last, none}
Average Range: {first, last, none}
Default: {none}

GivenNameAnchorFactor
Possible Settings: {0.00, 0.01,... 1.00}
Average Settings: {.50...70}
Default: {.70}

8.28 GivenNameCheckTAQ

When the GivenNameCheckTAQ = "off", no TAQ processing will take place at all.

When the GivenNameCheckTAQ = "remove", then TAQ(s) will simply be removed from the name data.

When the GivenNameCheckTAQ = "score", TAQ(s) will be identified, removed, and associated with each relevant name segment during preprocessing, and then the GivenNameTAQDeleteFactor, GivenNameTAQDeleteAbsentFactor, GivenNameTAQDisregardFactor, and GivenNameTAQDisregardAbsentFactor, will be multiplied against the GivenNameSegmentScore, which will in effect reduce the value of the GivenNameSegmentScore.

GivenNameCheckTAQ
Possible Settings: {off, remove, score}
Default: {score}

8.29 GivenNameMode

GivenNameMode operates exactly the same way as the SurnameMode but it applies to given names rather than surnames.

GivenNameMode:
Possible Settings: {highest, average, lowest}
Average Range: {highest, average, lowest}
Default: {average}

8.30 GivenNameExactInitialMatchScore

If GivenNameCheckInitial is set to True, then the GivenNameExactInitialMatchScore is used to indicate whether two single characters that match one another should be considered "exact matches", and therefore be assigned a score of 1.0. In some cases, it may be desirable to not consider two single characters as an exact match since it is possible that the two characters may represent two different names. In these cases, one might want to set the GivenNameExactInitialMatchScore = (1-GivenNameInitialScore)/2.

GivenNameExactInitialMatchScore
Possible Settings: {0.00, 0.1, ... 1.00}
Average Settings: {1.0}
Default: {1.0}

8.31 GivenNameInitialScore

The GivenNameInitialScore deals with the treatment of initials during a name check. In the following example, the initial "M" in the candidate could correspond to the name "Mohamed" in the query. Instead of considering it as a single digraph match, which in this case, would yield a score of .125, the "M" is given the value of the GivenNameInitialScore.

GivenNameInitialScore : .85
Query : Ali, Mohamed
Candidate : Ali, M

If a NameThreshold is defined, raising the GivenNameInitialScore will result in the return of more good hits since the value of an initial in a potential hit has been raised. Likewise, lowering the GivenNameInitialScore will result in a decrease in the number of hits returned.

GivenNameInitialScore
Possible Settings: {0.00, 0.01, ... 1.00}
Average Settings: {.60....90}
Default: {.85}

8.32 GNV-SCORE

The GNV-SCORE is the value given to a pair of Given Name variants found in the GIVEN-NAME-VARIANT Table. The GNV-SCORE is generally set very high, usually at .95.

GNV-SCORE
Possible Settings: {0.00, 0.01, ... 1.00}
Default: {defined by variant pair}

8.33 GivenNameOutOfPositionFactor (previously known as GNOOPS)

The GivenNameOutOfPositionFactor factor operates in the same manner as the SurnameOutOfPositionFactor.

GivenNameOutOfPositionFactor:
Possible Settings: {0.00, 0.01,... 1.00}
Average Range: {.50...70}
Default: {.55}

8.34 GivenNameTAQDisregardAbsentFactor

absent GN Disregard TAQ score – refer to section on TAQ scoring in main document for description.

GivenNameTAQDisregardAbsentFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.80}

8.35 GivenNameTAQDeleteAbsentFactor

absent GN Delete TAQ score – refer to section on TAQ scoring in main document for description.

GivenNameTAQDeleteAbsentFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.90}

8.36 GivenNameTAQDeleteFactor

delete GN TAQ score – refer to section on TAQ scoring in main document for description.

GivenNameTAQDeleteFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.85}

8.37 GivenNameTAQDisregardFactor

disregard GN TAQ score – refer to section on TAQ scoring in main document for description.

GivenNameTAQDisregardFactor
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.7}

8.38 FirstNameUnknownScore

If one of the comparands has been identified as having "first name unknown", then the segment score assigned when comparing that comparand with another is the FirstNameUnknownScore.

FirstNameUnknownScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.6}

8.39 NoFirstNameScore

If one of the comparands has been identified as having "no first name", then the segment score assigned when comparing that comparand with another is the NoFirstNameScore.

NoFirstNameScore
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.65}

8.40 GivenNameCompressedScore

In some instances, TAQ values become conjoined with stems in unpredictable ways. In some instances, two given name comparands are exact matches except for spacing (e.g., "nur al din" and "nuraldin"). If this is determined to be the case, the tpol will assign the GivenNameCompressedScore to the GivenNameScore.

GivenNameCompressedScore_
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.9}

8.41 GivenNameThreshold (previously known as GNTHRESH)

The GivenNameThreshold is the threshold which the GivenNameScore must exceed in order for the candidate name to be included in the Results list. If a developer wants to define a threshold rather than return the TOP X names, then this parameter may be set to some value other than 0. Setting the GivenNameThreshold to 0 essentially turns off the GivenNameThreshold. As the GivenNameThreshold is raised, fewer candidate names will be returned as it will be more difficult for a candidate name to pass the higher GivenNameThreshold. Conversely, as the GivenNameThreshold is lowered, more candidate names will be returned as it will be easier for a candidate name to pass the lower GivenNameThreshold.

GivenNameThreshold
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.50}

8.42 GivenNameWeight

The GivenNameWeight is the factor (weight) that can be applied to the GivenNameScore when determining whether a candidate name is to be included in the Results list. This weight factor enables one to assign more or less emphasis to a potential candidate based on the GivenNameScore. The higher the GivenNameWeight, the greater the value of the GivenNameScore contribution to the overall NameScore. If the GivenNameWeight is set to 0, the GivenNameScore will not contribute any value to the overall NameScore. In Version 1, the exception to this occurs if the SurnameWeight is also set to 0, in which case, the weight factors cancel one another out. In Version 1, we multiply the GivenNameScore by the GivenNameWeight as part of the default overall NameScore calculation. Note that developers may or may not choose to apply the GivenNameWeight when calculating an overall NameScore if they create a different scoring algorithm.

GivenNameWeight
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.80}

8.43 NameThreshold

The NameThreshold is the threshold which the NameScore must exceed in order for the candidate name to be included in the Results list. If a developer wants to define a threshold rather than return the TOP X names, then this parameter may be set to some value other than 0. Setting the NameThreshold to 0 essentially turns off the NameThreshold. As the NameThreshold is raised, fewer candidate names will be returned as it will be more difficult for a candidate name to pass the higher NameThreshold. Conversely, as the NameThreshold is lowered, more candidate names will be returned as it will be easier for a candidate name to pass the lower NameThreshold.

NameThreshold
Possible Settings: {0.0, 0.1, ...1.0}
Default: {.60}

Name Field	Parameter	Valid Range / Set of Values	Generic Default Value	Anglo Default Value (Eng/US)	Arabic Default Value (Egypt)	Chinese Default Value (China)	Hispanic Default Value (Mexico)	Korean Default Value (Korea)	Russian Default Value (Russia)
SN	SurnameCheckInitial	{T, F}	F	F	T	F	T	F	T
SN	SurnameCheckVariant	{T, F}	T	T	F	T	T	T	F
SN	SurnameCheckBias	{T, F}	F	F	F	F	F	F	T
SN	SurnameCheckUnknownNotExist	{T, F}	F	F	F	F	F	F	F
SN	SurnameCheckCompressed	{T, F}	F	F	T	F	T	F	F
SN	SurnameAnchorSegment	{first, last, none}	none	none	none	none	first	none	none
SN	SurnameCheckTAQ	{off, remove, score}	score	score	score	score	score	score	score
SN	SurnameMode	{highest, average, lowest}	average	average	average	average	average	average	average
SN	SurnameExactInitialMatchScore	{0.0, 0.1, ...1.0}	1.0	1.0	1.0	0	1.0	0	1.0
SN	SurnameInitialScore	{0.0, 0.1, ...1.0}	0	0	.85	0	.85	0	.85
SN	SNV-Score *	{0.0, 0.1, ...1.0}	-	-	-	-	-	-	-
SN	SurnameOutOfPositionFactor	{0.0, 0.1, ...1.0}	.60	.60	.90	1.0	.60	.63	.80
SN	SurnameAnchorFactor	{0.0, 0.1, ...1.0}	0	0	0	0	.70	0	0
SN	SurnameTAQDisregardAbsentFactor	{0.0, 0.1, ...1.0}	.80	.80	.80	.80	.80	.80	.80
SN	SurnameTAQDeleteAbsentFactor	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
SN	SurnameTAQDeleteFactor	{0.0, 0.1, ...1.0}	.85	.85	.85	.85	.85	.85	.85
SN	SurnameTAQDisregardFactor	{0.0, 0.1, ...1.0}	.70	.70	.70	.70	.70	.70	.70
SN	LastNameUnknownScore	{0.0, 0.1, ...1.0}	.60	.60	.60	.60	.60	.60	.60
SN	NoLastNameScore	{0.0, 0.1, ...1.0}	.65	.65	.65	.65	.65	.65	.65
SN	SurnameCompressedScore	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
SN	SurnameThreshold	{0.0, 0.1, ...1.0}	.50	.50	.63	.70	.60	.63	.62
SN	SurnameWeight	{0.0, 0.1, ...1.0}	1.0	1.0	.80	1.0	1.0	1.0	1.0
GN	GivenNameCheckInitial	{T, F}	T	T	T	F	T	F	T

LAS NameCheck Default Parameters

GN	GivenNameCheckVariant	{T, F}	T	T	F	T	T	T	F
GN	GivenNameCheckBias	{T, F}	F	F	F	F	F	F	F
GN	GivenNameCheckUnknownNotExist	{T, F}	F	F	F	F	F	F	F
GN	GivenNameCheckCompressed	{T, F}	F	F	T	F	T	F	F
GN	GivenNameAnchorSegment	{first, last, none}	none	none	none	none	none	none	first
GN	GivenNameCheckTAQ	{off, remove, score}	score	score	score	score	score	score	score
GN	GivenNameMode	{highest, average, lowest}	average	average	average	lowest	average	average	highest
GN	GivenNameExactInitialMatchScore	{0.0, 0.1, ...1.0}	1.0	1.0	1.0	0	1.0	0	1.0
GN	GivenNameInitialScore	{0.0, 0.1, ...1.0}	.85	.85	.85	0	.85	0	.85
GN	GNV-Score *	{0.0, 0.1, ...1.0}	-	-	-	-	-	-	-
GN	GivenNameOutOfPositionFactor	{0.0, 0.1, ...1.0}	.60	.60	.70	0	.60	.69	.65
GN	GivenNameAnchorFactor	{0.0, 0.1, ...1.0}	0	0	0	0	0	0	.60
GN	GivenNameTAQDisregardAbsentFactor	{0.0, 0.1, ...1.0}	.80	.80	.80	.80	.80	.80	.80
GN	GivenNameTAQDeleteAbsentFactor	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
GN	GivenNameTAQDeleteFactor	{0.0, 0.1, ...1.0}	.85	.85	.85	.85	.85	.85	.85
GN	GivenNameTAQDisregardFactor	{0.0, 0.1, ...1.0}	.70	.70	.70	.70	.70	.70	.70
GN	FirstNameUnknownScore	{0.0, 0.1, ...1.0}	.60	.60	.60	.60	.60	.60	.60
GN	NoFirstNameScore	{0.0, 0.1, ...1.0}	.65	.65	.65	.65	.65	.65	.65
GN	GivenNameCompressedScore	{0.0, 0.1, ...1.0}	.90	.90	.90	.90	.90	.90	.90
GN	GivenNameThreshold	{0.0, 0.1, ...1.0}	.50	.50	.63	.70	.60	.69	.60
GN	GivenNameWeight	{0.0, 0.1, ...1.0}	.80	.80	1.0	.80	.80	.80	.80
SN+GN	NameThreshold **	{0.0, 0.1, ...1.0}	.60	.60	.63	.70	.60	.66	.61

(Note that the values of the parameters that exist in DNC were mapped into their related parameter value in this set of default values. All new parameters were assigned a "best guess" value at this point. Some adjustments were made to the DNC GNOOPS + SNOOPS parameters).

* SNV-Score and GNV-Score values are not included in this table since the scores are actually associated with a specified variant pair, and are contained in the SURNAME-VARIANT and GIVEN-NAME-VARIANT Tables, respectively. The developer can not override the SNV-Score and GNV-Score through the API. Changes to these scores must be made through a separate VariantManager utility. Refer to the sections on SN Variants and GN Variants for more details.

** NameThreshold was calculated by scoring $\frac{(SurnameThreshold*SurnameWeight)+(GivenNameThreshold*GivenNameWeight)}{(SurnameWeight+GivenNameWeight)}$

Language Analysis Systems, Inc.

SearchSuite's™

NameHunter™

Developer's Documentation

Table of Contents

1. Introduction

The Social Security Administration (SSA) is seeking vendors who can provide software that can be used to build and access a file/data base using (customer) name as the access key. It will be used to retrieve information from a file where the names are often incomplete/truncated, names with unusual construct, or many times misspelled. It will also be used to match transaction records against a master name file/database. The software must also be able to evaluate (score/rate) the strength of one name string against another, both in on-line and batch processing. The software should also be flexible as to the data store in which the names are stored, giving SSA flexibility as to the storage vehicle.

Language Analysis Systems, Inc. (LAS), offers a set of Application Programming Interfaces (APIs) that enhance automated solutions to name searching issues by internalizing knowledge about cultural variation in names.

LAS implements a multifaceted approach to multicultural name searching. For example, in the Hispanic culture, an individual typically has a compound family name (e.g., *Aranxta SANCHEZ VICARIO*), the first of which (*SANCHEZ*) provides the more valuable identifying information. In contrast, although Portuguese names also typically have compound family names and look very similar to Hispanic names (e.g., *Maria FERREIRA DOS SANTOS*), the second family name (*DOS SANTOS*) provides the more valuable identifying information. If a single solution were proposed – where, for example, the *Last Name* is the important name, as in American names – Hispanic names would not be adequately accommodated.

The LAS solution applies whatever resources will adequately address the problem at hand whether the variation is cross-cultural or arises from spelling variation, from transcription from other writing systems, from sound similarity, or from missing or additional information.

Spelling Variations. Spelling variations can usually be addressed via character-matching techniques (e.g., *LESLEY, LESLIE*). However, false positive matches can easily result from traditional string or character comparisons when morphological endings such as OVIC occur at the end of a name (e.g. *ZELENOVIC, JOVANOVIC*).

Transcription Issues. Transcription variation generates a unique set of issues that result from different character sets, dialectal variations, and sounds that are not duplicated in Roman script. A single Chinese character (ideogram) can be transcribed to produce numerous Roman forms that have little or no resemblance to one another due to dialectal variations. For example, few individuals would recognize that *CHANG, JANG* and *ZHANG* are different representations of the exact same Chinese name, 張.

Sound Similarity. Names are often misheard or misrepresented as a result of pronunciation and expected spelling. *WOOSTER, WORCHESTER*, and *WUSTER* may or may not be pronounced identically and depending on the pronunciation, an individual hearing the name may expect a certain spelling representation. When sharing name data orally, both the pronunciation by the speaker and the expectations of the listener may have significant impact on the final representation of a name in a database system or written form.

Missing or Additional Data. Another common cause of name variation is the inclusion or exclusion of name data. Depending on the data source, names may be formal such as *THOMAS EDWARD WINTHROP III*, or informal such as *TOM WINTHROP*. A name search system must be capable of relating these two names to one another regardless whether all or some portion of the name is available. Note that missing or additional data may include valuable name segments (e.g., *EDWARD* in the example above), or less pertinent information such as titles, prefixes, suffixes, or qualifiers (e.g., the qualifier *III* in the example above).

A single solution cannot address the range of problems posed by multicultural name searching. Neither the sound similarity in names such as *SHAWN SMYTHE* and *SEAN SMITH*, nor the transcription variation in *IMHEMED BUCHLEIBI* and *MOHAMMED ABU SHLAYBY* can be easily handled by *character-matching techniques*. The differences are too great. Many search systems attempt to address these difficulties with *equivalency lists or tables*. While such lists can accommodate some of the most common variations, they are exceptionally limited, especially when it comes to random variation or error.

Keyed retrieval – using Soundex-like keys, for example – may be able to level some of the differences, but most keys are based on variations found in English, and therefore, do not accommodate the variation typical of other languages; nor do they accommodate random errors. For example, a standard Soundex key on the name *DOESCHER* would be D226; for the similar name *DOERSHER*, the key would be D626. Because the keys do not match, retrieval of these similar names would **NOT** take place.

The LAS Suite of Tools supplies the techniques necessary for complete and accurate retrieval of person, organization, and place name information. The LAS Suite of Tools is grounded in exacting cultural analysis and research, provides a broader and deeper search, and accommodates random variation.

WorldSearch™ (referred to as SNAPI in the enclosed documentation) employs multiple evaluation techniques to evaluate and score similar data. This tool determines whether two names are similar and assigns a score indicating the probability that the two names are in fact variations of one another. The tool incorporates information regarding variations in spelling, discrepancy in the amount of information included, exclusion of expected information, and positional information in order to establish a name score, which indicates the probability that the two names represent the same individual. The tool also orders scored similar data based on proximity rules. For example, an exact match should always appear at the top of any ordered match list. Other variations are ordered based on variations in spelling, inclusion of additional information, exclusion of expected information, and positional information.

WorldSearch™ can be used to match transaction records against a master name file/database. It can also be used to evaluate (score/rate) the strength of one name string against another, both in on-line and batch processing. **WorldSearch™** is totally flexible as to the data store in which the names are stored, thus providing SSA flexibility in their selection of the data store. **WorldSearch™** is extremely flexible and extensible; supporting more than 40 tune-able parameters, and the inclusion of additional data elements in the scoring mechanism, as well as modification to the actual scoring mechanism itself to accommodate customer-specific needs.

Prime Contact:
Leslie Minnix-Wolfe - Director of Technical Development
Language Analysis Systems, Inc.
2214 Rock Hill Road
Herndon, VA 20170
Phone:
(703) 834-6200 x229
Fax:
(703) 834-6230
Email:
lmw@las-inc.com

Reference Information

Jerry Cuffee, Office of Research and Development, (703) 613-8758

Sam Whitmer, US Department of State, (202) 663-1102

Jim Richardson, (703) 893-0427

2. Mandatory requirements of the software package must be:

2.1 Field proven with a proven track record, currently commercially available, in use in production environments at multiple customer sites, SSA must be able to contact existing users; software that is in BETA testing or in development is not acceptable;

Earlier versions of **WorldSearch™** are fielded at over 210 consular sites around the world in support of the US Department of State. The latest, more advanced, version of **WorldSearch™** is commercially available today.

2.2 Tunable/flexible in its ability to create data base keys for storing records in the creation and updating of the data base. Allow various ways to develop an access key to search the data base in order to retrieve data by a client's name.

The current version of **WorldSearch™** does not create data base keys for accessing records in a data base. Keyed retrieval is inflexible by definition. For example, existing sound-based keyed retrieval methods, such as Soundex and NYSIIS (a derivative of Soundex) are very limited solutions. Using these keying techniques, two different names generate the same key, and therefore would be retrieved together:

Name	Soundex Key	NYSIIS Key
SMOOT	S530	SNAT
SMITH	S530	SNAT

More importantly, the same name spelled two ways has two different codes, and therefore would not be retrieved together:

Name	Soundex Key	NYSIIS Key
WUSTER	W236	WASTAR
WORCHESTER	W622	WARCASTAR

Random errors and truncations are real problems for these and other keying strategies. Existing keyed-retrieval systems address only one aspect of the name search problem, thereby eliminating the possibility of returning valid matches. They provide a one-size-fits-all approach to a much more complex problem. Their approach also tends to be Anglo-centric, which inhibits one's ability to address the issues of multi-cultural names, such as Hispanic, Arabic, and American Indian names. Keys in general cannot accommodate random variation and extreme spelling variations. **WorldSearch™** promotes multiple data base sub-setting strategies to work around the limitations inherent in keyed retrieval. These strategies incorporate other data elements, as appropriate and incorporate additional information about the name, such as the cultural/ethnic origin.

Future versions of **WorldSearch™** will incorporate among other features, sophisticated phonetic indexes as well as enhanced pre-processing of data to accommodate extreme spelling variations prior to index generation. Culture-specific indexing strategies will be incorporated to accommodate different cultural issues as well as the random errors that are concealed by sound-based keyed retrieval techniques like Soundex and its derivatives. Note that current plans for **WorldSearch™** indexes include keyed indexes as well as non-key based indexing (e.g., bitmap indexing). An initial offering of a single indexing (keyed or non-keyed) strategy should be commercially available in the first quarter of 1998.

2.3 Allow adjustments to be made to the scoring mechanism. Ideally these changes should be done via initialization files, rather than package source code changes (which would result in customized variations of the original product). Allow for scoring based on the name string and Social Security Number string.

Flexibility and extensibility are two of the principles upon which **WorldSearch™** was developed. There are over 40 tune-able parameters provided to enable adjustments in the scoring mechanism. In addition, culture-specific packages of parameters are provided with the tool to facilitate culture-specific handling of name issues. Applications can be constructed to enable the end-user to make adjustments in an interactive mode or can override the default parameter settings to accommodate customer-specific requirements in support of either or both the interactive mode and a batch mode. For example, a batch process might be established to compare two names using a "tight" search, and if no matches are found, a subsequent process might be established to then compare the two names using a "loose" search. Consider the following:

	Given name	Surname
Query:	Gerald	David

Data record: David Gerald

A "tight" search might be defined as one that considers the names in their specific surname and given name format, and a "loose" version of that search, might consider inversion of the surname and given name. For more details, refer to the Developer's Documentation on the SNQueryParms Class.

Additional data elements may also be integrated into the scoring mechanism to accommodate scoring based on name data as well as any other desired data elements, such as Social Security Number. For more details, refer to the Developer's.Documentation on the SNQueryNameData Class and SNEvalNameData Class.

2.4 Allow for transposed numbers and transposed letters.

In addition to predictable variations in names, **WorldSearch™** easily handles unpredictable variations such as transposed letters (e.g., RODRIGUEZ = RODIGRUEZ), as well as other random errors, such as truncation (e.g., CORNWALL = CORNW) and typos (e.g., BOMEZ = GOMEZ).

2.5 Allow nicknames and derivative names to be scored as equal (e.g. Anthony = Tony, Jose = Joseph = Joey = Giuseppe, etc.). T he package should have a built in store of nicknames, derivatives and it should allow for customization of the nicknames and derivatives. Allow equating names such as St. = Saint.

In addition to names with predictable similar spelling variations (e.g., *GONZALEZ = GONZALES*), **WorldSearch™** provides for very sophisticated handling of:

- predictable similar sounding, but different spelling variations (e.g., *CRUZ = KRUSE = CREWS = CRUISE*);
- nicknames (e.g., *ANTHONY = TONY*);
- abbreviations (e.g., *SAINT = ST.*);
- gender differences (e.g., *MARIA = MARIO*);
- morphological endings (e.g., *JOHNS = JOHNSON*); and
- other derivative names.

WorldSearch™ differentiates between the different types of variations that occur and therefore, does not simply score two variations as equal. Rather, it provides a finer level of granularity in determining the degree of similarity between two name variations. As a result, customization of these variations is not provided with the current version of the tool, as it requires rather extensive knowledge of name searching. Future versions of the tool may allow for customization, however.

WorldSearch™ provides culture-specific sets of these values in order to handle cross-cultural issues. For example, *VAN* might be considered a nickname for *VANESSA* or *VANYA*, but it is also considered a prefix in Dutch names like *VAN ROSSUM*, and a gender marker in Vietnamese names like *VAN NGUYEN*. Therefore, one might not want to consider VANESSA

NGUYEN as a match for *THANH VAN NGUYEN*, depending on the nature of the data. For more details, refer to the Developer's Documentation on Variant processing.

2.6 Allow for "cleansing (ignore) titles (e.g. Dr., Jr., RN, Sr., Etc.).

WorldSearch™ provides for very sophisticated handling of titles (e.g., *DR., MR.*) affixes (prefixes (e.g. *DE, LA, VAN*), suffixes (e.g., *Aldin, Din*)), and qualifiers (e.g., *JR., SR., RN*). **WorldSearch™** provides culture-specific sets of these values in order to handle cross-cultural issues. For example, *BEN* is a common prefix in Arabic names like *BEN GURION*, but it is also a common given name or nickname (e.g., *BENJAMIN = BEN*) in Anglo cultures. **WorldSearch™** does not simply ignore TAQ values, as in some cases, these values provide additional information when evaluating a candidate name. For example, if one is searching for *RICHARD ANTON UHRIG, JR.* and finds *RICHARD ANTON UHRIG, SR.*, depending on the application, the *Jr.* and *Sr.* provide information that is valuable in determining whether these two records match or not. **WorldSearch™** provides the flexibility to decide how and when to apply these more sophisticated scoring techniques. TAQ processing can be turned off entirely, or turned on to simply ignore all TAQ values, or to score the TAQ values. For more details, refer to the Developer's Documentation on TAQ processing.

2.7 Able to run on an IBM MVS/ESA compatible mainframe. "Callable" from batch or CICS/COBOL.

WorldSearch™ is composed of one or more C++ APIs and is compatible with any modern platform with a C++ compiler. There are several ways of accomplishing this, but one of the better approaches is to establish a Name Server which receives search requests from an application, processes the request, and then returns the desired results to the calling application. This approach provides more flexibility and extensibility to the Name Server to support multiple application interfaces such as on-line versus batch. It also eliminates the need to have a COBOL application become a COBOL/C application, which is clearly more complex to develop as well as more difficult to maintain.

2.8 Work with a multi-segmented data base (containing millions of records per segment). The entire data base currently contains in excess of 200 million records.

WorldSearch™ is entirely independent of the data store, and therefore, can work with a multi-segmented data base. Different strategies can be implemented to handle the large volume of data.

2.9 Contain name match profiling/tuning/evaluation software as part of the suite of tools.

WorldSearch™ is essentially a name match profiling/tuning/evaluation tool. It provides the capability to evaluate and score name data. It also provides complete flexibility to tune the evaluation mechanism and extensibility to incorporate additional information into the evaluation algorithm(s).

In addition, consulting services are available to provide more extensive analysis and profiling of the data, and subsequent tuning of the parameters and scoring techniques, as well as the incorporation of application-specific requirements.

2.10 The "support package" must contain full documentation that is currently available, pre-existing training programs and courses, customer support for immediate consultation on technical problems/issues (NN hours per day, from xx-yy).

A full copy of the latest version of the ***WorldSearch***™ Developer's Documentation is included with this RFI response. This documentation is provided in HTML format and will soon be available via the LAS web page.

A maximum of 40 hours of technical support is included with the base purchase price of the product to assist with the initial understanding and use of the API's.

With the purchase of an annual maintenance agreement, technical support is provided 24 hours per day, 7 days a week. Technical support will provide on-going consultation to address technical problems/issues with the integration and use of the APIs.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

WELCOME

Welcome to the LAS SNAPI Product Support Site. The purpose of this web site is to enhance the support services we provide to our customers. We have provided a number of resources here to help you resolve problems, report bugs, and suggest improvements to our products and services.

You may also obtain technical support via either :

Telephone: (703) 834-6200

E-mail: snapi@las-inc.com.

● Welcome	This Page.
● Overview	General Description of SNAPI From A Developer's Perspective. A good place to start.
● What's New	News & Announcements.
● FAQs	Answers To Frequently Asked Questions.
● Tutorial	Detailed explanation of how to write applications using SNAPI.
● API Documentation	Full API Documentation.
● Sample Code	Source Code Illustrating SNAPI's Most Important Features.
● Bugs	List Of Known Problems.
● Suggestions	Tell Us The FeaturesYou Would Like To See In Our Next Version.
● Download	Download Source Code, Sample Applications, Demo Versions.
● Search	Search the entire SNAPI Site.

SNAPI is a trademark of Language Analysis Systems. All other products mentioned are registered trademarks or trademarks of their respective companies.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday November 21, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

This section gives a basic and brief overview of the SNAPI system. For a more detailed explanation of how to use the API, please see the Tutorial. Once you have read the overview, click here for some suggestions on where to go from here.

Developer's Overview

SNAPI (Smart Name API) is a set of programming libraries (functions and classes) that enables a developer to add fuzzy personal name searching to an application. It gives you the capability to perform operations such as "Give me the 10 closest names to 'James Slesinger' from my database", or "Give me all the names from my database that match 'John Wong' with a degree of confidence of 0.9" or "Tell me the degree of similarity between 'Paul Vanesann' and 'P Vanlesann'". The system uses a variety of linguistic techniques to achieve solid, dependable results.

The libraries are coded in C++, and can be easily integrated into any application written in C++. SNAPI is available on any platform that supports a C++ compiler. The SNAPI system was designed with the following goals: simplicity and ease of integration, maximum flexibility, and maximum extensibility.

Simplicity and Ease of Integration

From the developer perspective, the SNAPI system is quite simple. A typical name search requires the use of just four classes (SNQueryParms, SNQueryNameData, SNEvalNameData, and SNResultsList). In addition, the extra code required to integrate SNAPI is minimal. Both the code snippet in the Tutorial, and the code samples illustrate this point.

SNAPI's interface is simplified by the fact that it makes no assumptions about your data and how it is stored. The philosophy behind our product is that you know your data better than anyone else. This allows for a much cleaner design - You provide the name you are looking for, as well as the names from your database. The product tells you which names are likely matches, and qualifies their degree of similarity. Behind the scenes, the process is much more complex, but from the perspective of the developer, the tool appears straight-forward and easy to integrate.

Flexibility

Searches via the SNAPI system are configurable by adjusting any of 43 parameters. Each parameter controls some aspect of how two names are evaluated when determining if they are similar. Some of the more basic parameters set thresholds for determining how close two names must be in order to be considered a match. Other parameters control more complex processing, such as how to handle multi-segment names. In general, only a small set of parameters need to be adjusted by the developer, because reasonable defaults exist for each one. Documentation for the SNQueryParms class discusses each of the parameters.

SNAPI also provides pre-defined packages of parameters, each tailored to a particular culture or ethnicity. For example, Hispanic names have certain characteristics such as compound surnames (e.g., Torres de la Cruz) that can cause problems when searching for Hispanic names using conventional methods, which are typically Anglo-centric. The Hispanic parameters package contains settings that address Hispanic-specific name issues. New cultural/ethnic parameter packages can be established and existing packages can be modified as desired. The SNQueryParms constructor describes the various parameter packages available.

Extensibility

Because SNAPI is a C++ object framework, developers can extend the existing functionality to incorporate additional data elements in the scoring algorithm or create evaluation methods specific to their business or application needs. For example, a database might contain Social Security Number in addition to given name and surname. SNAPI only provides for comparisons of name data. However, a developer can take advantage of class inheritance (a feature of C++), and easily subclass SNAPI's SNEvalNameData and SNQueryNameData objects to include SSN or any other desired data element(s). This data can then be used in the methods that score evaluation names, and determine which evaluation names are matches. In other words, record matching can be performed using name data in conjunction with other available data element information.

Developers can also provide custom methods for determining if an evaluation name matches a query name or not. SNAPI's default method compares the average of the given name score and surname score to a developer supplied threshold value. However, a more complex method may be desired. For example, the business rules of an application might dictate that a name can not be considered a match unless either the surname or given name is an exact match. By overriding SNAPI's default method, the developer can easily implement this logic in just a few lines of code.

Where To Go From Here

Now that you have a basic understanding of what the SNAPI API provides, we recommend proceeding to the tutorial. There, you will find several "code snippets" that demonstrate how to use the SNAPI objects. From there, you can reference the API documenation for a more detailed discussion of the classes and methods. Alternatively, you can view the FAQ lists to search for the answer to a particular question.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday November 21, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNEvalNameData Class

Overview

A subclass of SNNameData, SNEvalNameData represents a candidate name that will be compared against a query name (an SNQueryNameData object). Built on top of SNNameData, it adds the data necessary to keep track of scores resulting from a comparison. In addition, it adds a method to perform a comparison between itself and an SNQueryNameData object.

The developer may subclass this class to add any data that might be useful to attach to a candidate name. This practice becomes important if the developer is using an SNResultsList to manage the hits during a session, because they will probably need some unique way (within their system) to identify the SNEvalNameData objects that end up in the results list after all names have been evaluated. A common example is a subclass that adds a database recordId field. Once all candidate name objects have been processed, the results list can be queried to obtain those objects that are considered matches. Each object that was considered a match can then be queried to obtain its database recordId.

The developer is responsible for deleting any SNEvalNameData created by their code. Typically, the developer will construct a new SNEvalNameData object, compare it to the query name (an SNQueryNameData object), and then delete the SNEvalNameData object. Before deleting the object, the developer may wish to examine the scores that result from the comparison (e.g. getNameScore(), getSnScore(), etc.). If an SNResultsList object is being used in the query process, the developer can safely delete the SNEvalNameData object after the comparison, because the SNResultsList object makes copies of the objects it manages. See the SNResultsList documentation for a more detailed discussion.

Subclassing

Developers may wish to subclass SNEvalNameData for a variety of reasons. The most common need for subclassing is to allow application specific data to be attached to each evaluation name. For example, an application might read candidate names from a database, where each name consists of a given name, surname, unique record id, and birthdate. However, SNEvalNameData only knows about given name and surname, and is oblivious to record Id and birth date. By subclassing SNEvalNameData, a developer can add these or any other data elements. This method of tagging candidate name objects becomes important when an SNResultsList is used to manage hits. In this case, some method is needed to link the objects returned from the SNResultsList back to their associated data in the original data source.

Subclassing can also allow an application to use extra data to affect the search and evaluation process. Continuing the example above, suppose a developer wants to include birth date as a factor in the search. The developer, having subclassed SNEvalNameData, can override the calcNameScore() method to include age differential (how close in age the candidate is to the query) in the evaluation method.

The following are methods that can be overridden to provide specialized processing in a subclass:

~SNEvalNameData() Destructor for the class. Ensure that the destructor for your subclass frees any resources your subclass allocates.

calcComponentScores() Calculates the name field (given name and surname) scores for an evaluation name. Override this if you wish to calculate scores using application-specific data.

calcNameScore() Determines the composite name score for an evaluation name. The composite score incorporates the component scores into a single value. Override this if you wish to incorporate application-specific data into the name score calculation.

compareScore() Compares two scored evaluation names. Override this if you wish to change the way evaluation names are sorted within an SNResultsList.

getCompResult() Determines if a scored evaluation name is a match or not. Override this if you wish to incorporate application-specific data into the "match/no match" decision process.

resetScores() Resets scores within the class. Override this if you are pre-loading evaluation names and have added additional score variables that need to be reset before performing a comparison.

Methods Summary

Common Methods:

SNEvalNameData() Various constructors for the class.

getGnScore() Returns the given name score after a comparison.

getNameScore() Returns the composite name score after a comparison.

getSnScore() Returns the surname score after a comparison.

performComp() Compares this object to an SNQueryNameData object.

Specialized Methods:

calcComponentScores() Overridable. Calculates component scores for an evaluation name. Not called directly by the developer.

calcNameScore() Overridable. Determines the composite name score for an evaluation name. Not called directly by the developer.

compareScore() Overridable. Compares this object to a second SNEvalNameData object after each has been compared to an SNQueryNameData object. Not called directly by the developer.

getCompResult() Overridable. Determines if this scored evaluation name object is to be considered a match. Not called directly by the developer.

resetScores() Overridable. Clears out the scoring information for this object, so that it can be used in a new query. Used by applications that load pre-processed representations of their database into memory for increased performance.

Attributes:

double gnScore; A value between 0.0 and 1.0 indicating how closely the given name matched during a comparison. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

double snScore; A value between 0.0 and 1.0 indicating how closely the surname matched during a comparison. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

double nameScore; A value between 0.0 and 1.0 indicating how closely the name (considered as a whole) matched during a comparison. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

int gnSegDifferential; The difference between the number of given name segments in this object's name and the SNQueryNameData object's name after a comparison. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

int snSegDifferential; The difference between the number of surname segments in this object's name and the SNQueryNameData object's name after a comparison. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

double gnSegScores[]; An array that holds the given name segment comparison scores for the name. This array is sometimes consulted by the compareScore() method. This attribute should never be altered by the developer. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

double snSegScores[]; An array that holds the surname segment comparison scores for the name. This array is sometimes consulted by the compareScore() method. This attribute should never be altered by the developer. This is a protected member, and can only be accessed directly by subclasses of SNEvalNameData.

Method Details:

Constructors:

```
□ SNEvalNameData(SNQueryParms *qParms, char *gn, char *sn);
□ SNEvalNameData(SNQueryParms *qParms, char *gn, char *sn, char *mn);
□ SNEvalNameData(SNQueryParms *qParms, char *name, SNNameFormat nameFormat);
```

Each constructor creates a new SNEvalNameData object. All forms of the constructor take a pointer to an SNQueryParms object, which should be the same pointer that was used to create the SNQueryNameData object that this object will be compared against.

The SNAPI system is based internally on a name model that considers given name and surname. However, other constructors are provided for cases where an alternate name model format is desired. In these cases, the constructor maps the supplied data into SNAPI's given name / surname model.

The first form of the constructor takes a given name and surname. This is the most efficient and accurate format, because the data already corresponds to SNAPI's internal name model.

The second form accommodates systems that have knowledge of a middle name. Currently, SNAPI maps the middle name into the given name. Future versions of SNAPI may provide more sophisticated handling of middle name data.

The third form accommodates systems that represent names as a single string, rather than separate fields. This form takes an SNNameFormat parameter that dictates how the string will be mapped into the given name / surname model. Values for the SNNameFormat parameter include:

SN_SURNAME_COMMA_GIVENNAME Expects the name string in the form "surname, given name". If no comma is found in the string, the given name is assumed to be unknown, and the entire string is placed in the surname field.

SN_LAST_SEG_IS_SURNAME Expects the string to be in the form "given_name surname". The last segment is placed in the surname field. All other segments are placed in the given name field. If a string has just one segment, that segment is placed in the surname field, and the given name field is assumed to be unknown. The processing is intelligent in that TAQ values are recognized when determining the last segment. This allows a name such as "Bob Jones Jr." to be correctly mapped with given name of "Bob" and a surname of "Jones Jr", rather than incorrectly assigning "Jr" as the surname.

SN_NAME_FORMAT_UNKNOWN Currently operates identically to SN_LAST_SEG_IS_SURNAME. Future versions of SNAPI might incorporate more sophisticated linguistic techniques to make automated decisions about parsing the string into the appropriate name fields.

Parameters:

`qParms` A pointer to a SNQueryParms object. The SNQueryParms object should be the same object that was used to create the SNQueryNameData object that this SNEvalNameData object will be compared against.

`gn` A NULL terminated string that represents the given name.

`sn` A NULL terminated string that represents the surname.

`mn` A NULL terminated string that represents the middle name.

`name` A NULL terminated string that represents all components of the name as a single string.

`nameFormat` An enumerated type value that specifies how to interpret the name string when breaking it into given name and surname. See documentation for SNNameFormat for valid values.

Return Values:

None.

Memory Management:

The responsibility of deleting an SNEvalNameData object lies with the developer. In general, an SNEvalNameData object should be deleted shortly after it has been compared to an SNQueryNameData object. The exception to this is an application that stores object representations of its search database in memory, and then reuses these objects for every query. In such cases, the SNEvalNameData objects should only be deleted once they are no longer needed (e.g. as the application exits).

Examples:

```
The example below shows four equivalent SNEvalNameData objects being
constructed. In each case, the supplied name gets mapped to an
internal representation of: Given Name = "Bob Earl", Surname =
"Jones".
```

```
SNEvalNameData   *candidate1;
SNEvalNameData   *candidate2;
SNEvalNameData   *candidate3;
SNEvalNameData   *candidate4;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);

candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
candidate2 = new SNEvalNameData(queryParms, "Bob", "Jones", "Earl");
candidate3 = new SNEvalNameData(queryParms, "Bob Earl Jones",
                                SN_LAST_SEG_IS_SURNAME);
candidate4 = new SNEvalNameData(queryParms, "Jones, Bob Earl",
                                SN_SURNAME_COMMA_GIVENNAME);

// delete the allocated objects somewhere below.
```

`double getGnScore();`

Returns the given name score calculated during the performComp() method. If called before performComp() is invoked, the result is undefined.

Most applications will call getNameScore(), which gives a score for the name as a whole. getGnScore() allows the developer to examine the given name separately, and is provided for those applications that require special consideration of the given name.

Parameters:

None.

Return Values:

A double value between 0.0 and 1.0, indicating how closely the given names of the query and candidate names matched.

Examples:

```
The example below shows a comparison and subsequent given name score
examination:
```

```
SNEvalNameData  *candidate;
SNQueryNameData *queryName;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
double          gnScore;
```

```
candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

candidate->performComp(queryName);

gnScore = candidate->getGnScore();
printf("Given Names Matched with a score of %f", gnScore);

  //    delete the allocated objects somewhere below.
```

```
SNReturnCode performComp(SNQueryNameData *queryName);
```

Compares this evaluation name object to a query name object (SNQueryNameData). On return from this method, score information can be retrieved from this object using methods such as getGnScore(), getNameScore(), etc. The comparison is conducted according to the parameters specified in the SNQueryParms object that was used to construct this SNEvalNameData object.

Parameters:

queryName A pointer to a SNQueryNameData object. This object is a representation of the query name, and should be constructed with the same SNQueryParms object that was used to create this SNEvalNameData object.

Return Values:

An SNReturnCode value indicating the result of the comparison. Values include SN_MATCH and SN_NO_MATCH, but the return code can also indicate a variety of errors. See the documentation for SNReturnCode for full details.

Examples:

The example below shows a sample comparison:

```
SNEvalNameData  *candidate;
SNQueryNameData *queryName;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode    retCode;

candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

retCode = candidate->performComp(queryName);
if (retCode == SN_MATCH)
   printf("Names Matched");
else
   if (retCode == SN_NO_MATCH)
```

```
      printf("No Match");
   else
      printf("Error!");

 //    delete the allocated objects somewhere below.
```

```
double getNameScore();
```

Returns the composite name score calculated during the performComp() method. The name score takes into account the given name score, the surname score, and the given name and surname weights. If called before performComp() is invoked, the result is undefined.

Parameters:

None.

Return Values:

A double value between 0.0 and 1.0, indicating how closely the query and candidate names matched.

Examples:

The example below shows a comparison and subsequent name score examination:

```
SNEvalNameData  *candidate;
SNQueryNameData *queryName;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
double          nameScore;

candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

candidate->performComp(queryName);

nameScore = candidate->getNameScore();
printf("Names Matched with a score of %f", nameScore);

//      delete the allocated objects somewhere below.
```

```
double getSnScore();
```

Returns the surname score calculated during the performComp() method. If called before performComp() is invoked, the result is undefined.

Most applications will call getNameScore(), which gives a score for the name as a whole. getSnScore() allows the developer to examine the surname separately, and is provided for those applications that require special consideration of the surname.

Parameters:

None.

Return Values:

A double value between 0.0 and 1.0, indicating how closely the surname(s) of the query and candidate names matched.

Examples:

The example below shows a comparison and subsequent surname score examination:

```
SNEvalNameData   *candidate;
SNQueryNameData  *queryName;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
double           snScore;

candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

candidate->performComp(queryName);

snScore = candidate->getSnScore();
printf("Surnames Matched with a score of %f", snScore);

//      delete the allocated objects somewhere below.
```

```
SNReturnCode performComp(SNQueryNameData *queryName);
```

Compares this evaluation name object to a query name object (SNQueryNameData). On return from this method, score information can be retrieved from this object using methods such as getGnScore(), getNameScore(), etc. The comparison is conducted according to the parameters specified in the SNQueryParms object that was used to construct this SNEvalNameData object.

Parameters:

`queryName` A pointer to a SNQueryNameData object. This object is a representation of the query name, and should be constructed with the same SNQueryParms object that was used to create this SNEvalNameData object.

Return Values:

An SNReturnCode value indicating the result of the comparison. Values include SN_MATCH and SN_NO_MATCH, but the return code can also indicate a variety of errors. See the documentation for SNReturnCode for full details.

Examples:

The example below shows a sample comparison:

```
SNEvalNameData  *candidate;
SNQueryNameData *queryName;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode    retCode;

candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

retCode = candidate->performComp(queryName);
if (retCode == SN_MATCH)
   printf("Names Matched");
else
   if (retCode == SN_NO_MATCH)
      printf("No Match");
   else
      printf("Error!");

  //    delete the allocated objects somewhere below.
```

```
virtual inline void calcComponentScores(SNQueryNameData *queryName)
```

Calculates the component scores for the evaluation name. This function is called by the API, not by the developer. Specifically, it is called by the performComp() method before the composite name score is calculated (via a call to calcNameScore()).

The method is virtual to allow subclasses of SNEvalNameData to provide score calculations for any application-specific data the developer may have added to the evaluation name. The default method calculates scores for the given name and surname components. Subclasses must call the base class implementation so that the given name and surname scores are set properly.

Parameters:

queryName A pointer to a SNQueryNameData object. This object is a representation of the query name, and should be constructed with the same SNQueryParms object that was used to create this SNEvalNameData object.

Return Values:

None.

Examples:

```
The example below shows a sample override of the
calcComponentScores() method. In the example, we have defined a
subclass of SNEvalNameData called MySNEvalNameData. This class
includes an SSN data member, and a Boolean ssnMatch flag that should
be set when the query and evaluation name have the same SSN.


void     MySNEvalNameData::calcComponentScores(SNQueryNameData *queryName)
{
   SNEvalNameData::calcComponentScores(queryName);       //   have to call t

   if (ssn == queryName->ssn)    {
      ssnMatch = TRUE;
```

```
        else
            ssnMatch = FALSE;
        }
    }
```

```
virtual void calcNameScore()
```

Calculates the composite name score, placing the result in the member variable nameScore. This function is called by the API, not by the developer. The method is virtual to allow subclasses of SNEvalNameData to incorporate other data and/or logic in the calculation.

The full implementation of SNEvalNameData::calcNameScore() appears in the SNEvalNameData.hpp header file. This gives the developer insight into how a customized method might incorporate additional data.

On exit from this function, the nameScore member variable should result with a value between 0.0 and 1.0.

Parameters:

None.

Return Values:

None.

Examples:

```
The example below shows a sample override of the calcNameScore()
method. In the example, we have defined a subclass of SNEvalNameData
called MySNEvalNameData. This class includes a bornYear member
variable. The sample calls the base class implementation, and then
gives special consideration to people born before 1900. A more
complicated example might replace the base class implementation
entirely.


void    MySNEvalNameData::calcNameScore()
{
   SNEvalNameData::calcNameScore();

   if (bornYear < 1900) {
      nameScore *= 1.1;     // give an extra 10 percent on the score.

      if (nameScore > 1.0)    // make sure we do not exceed a perfect score.
         nameScore = 1.0
   }
}
```

```
virtual int compareScore(SNEvalNameData *scoredName)
```

Compares the scored SNEvalNameData object to a second scored SNEvalNameData object. This function is called by the API, not by the developer. Specifically, it is called by an SNResultList object to determine the sort order of the matches it manages.

The method is virtual to allow subclasses of SNEvalNameDatat to incorporate other data and/or logic in the sorting process. In general, applications can override the calcNameScore() method to incorporate application-specific data into the calculation of the name score. The name score is the most important factor in the default sort method, so proper sorting occurs automatically. Because the compareScore() method is somewhat complex, overriding calcNameScore() is the preferred method.

However, there may be times when a more detailed modification of the sort method is required. For example, the developer may wish to introduce a data element that does not affect the name score at all, but does affect the sort order of any matches. In these cases, override the compareScore() method. The full implementation of SNEvalNameData::compareScore() appears in the SNEvalNameData.hpp header file. This gives the developer insight into how a customized method might incorporate additional data. The method is complex enough to warrant a brief discussion of its behavior (Discussion can also be found in the implementation of the function).

In general, the compareScore() method performs a series of comparisons to determine which evaluation name is better (i.e. closer to the query name). The comparisons occur in descending order of importance. If any comparison yields a discrepancy, the comparison stops there. Otherwise, we proceed to the next comparison. The order of comparisons is as follows:

```
nameScore,
snScore;
if (snSegmentScoreMode == HIGHEST)
   snSegmentScores
gnScore,
if (gnSegmentScoreMode == HIGHEST)
   gnSegmentScores
snSegDiff (the difference in the number of sn segments between the query and the
gnSegDiff (the difference in the number of gn segments between the query and the
```

A override of compareScore() would insert a comparison of some application-specific data at the desired point. For example, our subclass might include a Boolean flag indicating if this name's Social Security Number matched that of the query name exactly. Further, suppose our business rules dictate that all exact SSN matches should appear at the top of the results, regardless of name score. In this case, we would perform a comparison of the Boolean flag prior to checking the name score:

```
if (ssnMatch || scoredName->ssnMatch)   {
   if (!ssnMatch)
      return -1;   // the scoredName is better, since it's an exact SSN match
   else {
      if (!scoredName->ssnMatch)   // this name is better, since it's an exact S
        return 1;
   }
}

// proceed with rest of default comparison, since both were an exact SSN match,
```

In our contrived example, it would have been possible to just perform our check, and in the event of a tie, call the base class implementation. If our desired insertion point had been somewhere in the middle of the comparison order, we would be forced to provide a full version of the method. The example below demonstrates this.

Parameters:

scoredName A pointer to an SNEvalNameData object. This object is a representation of another evaluation name that has already been scored.

Return Values:

1 if this evaluation name is a better match than the supplied evaluation name.

-1 if the supplied evaluation name is a better match than this evaluation name.

0 if both names match the query name with the same degree of confidence.

Examples:

The example below shows an override of the compareScore() method. The example supposes a subclass of SNEvalNameData that introduces an integer variable, ssnScore, which is a number between 0 and 9, indicating how many digits matched between the query and evaluation name's SSN. Suppose we want the SSN score to be considered after the overall name score, but before a comparison of the surname and given name scores. The resulting method looks a lot like the base class implementation, but we have inserted a comparison of the ssnScore just after the nameScore comparison:

```
virtual int     MySNEvalNameData::compareScore(SNEvalNameData *scoredName)
   int  rc;
   double       scoreDiff = scoredName->getNameScore() - nameScore;

   if (scoreDiff < 0.0)
      rc = -1;
   else if (scoreDiff > 0.0)
      rc = 1;
   else {
      scoreDiff = scoredName->ssnScore - ssnScore;     //   <--  inserted c
      if (scoreDiff < 0.0)
         rc = -1;
      else if (scoreDiff > 0.0)
         rc = 1;
      else      {                                      //   <-- end of inse
         //     scores were the same, so look at snScore
         scoreDiff = scoredName->getSnScore() - snScore;
         if (scoreDiff < 0.0)
            rc = -1;
         else if (scoreDiff > 0.0)
            rc = 1;
         else    {
            //  see if our snSegmentScoreMode mode is
            //  HIGHEST. If it is, we need to check the sn segment scores
            if (queryParms->getSnSegmentScoreMode() == SN_SEGMODE_HIGHEST).
               scoreDiff = compareSegmentScores(scoredName, SN_LAST_NAME);

            //  see if we still are equal after the above check
            if (scoreDiff < 0.0)
               rc = -1;
            else if (scoreDiff > 0.0)
               rc = 1;
            else        {
               //       scores were the same, so look at gnScore
               scoreDiff = scoredName->getGnScore() - gnScore;
               if (scoreDiff < 0.0)
                  rc = -1;
```

```
              else if (scoreDiff > 0.0)
                 rc = 1;
              else      {
                 if (queryParms->getGnSegmentScoreMode() == SN_SEGMODE_HIG
                    scoreDiff = compareSegmentScores(scoredName, SN_FIRST_

                 //    see if we still are equal after the above check
                 if (scoreDiff < 0.0)
                    rc = -1;
                 else if (scoreDiff > 0.0)
                    rc = 1;
                 else  {
                    int        segDiff;

                    // scores were the same, so look at snSegDifferential
                    // for this case, smaller is better, so switch operand
                    segDiff = snSegDifferential - scoredName->snSegDiffere
                    if (segDiff < 0)
                       rc = -1;
                    else if (segDiff > 0)
                       rc = 1;
                    else       {
                       //      scores were the same, so look at snSegDiffe
                       //      for this case, smaller is better, so switch
                       segDiff = gnSegDifferential - scoredName->gnSegDiff
                       if (segDiff < 0)
                          rc = -1;
                       else if (segDiff > 0)
                          rc = 1;
                       else    {
                          rc = 0;
                       }
                     }
                  }
               }
            }
         }
      }
   }

   return rc;
}
```

```
virtual SNReturnCode getCompResult()
```

Determines if the SNEvalNameData object is considered a match or not. This function is called by the API, not by the developer. Specifically, it is called during the performComp() method after the name has been scored.

The method is virtual to allow subclasses of SNEvalNameData to incorporate other data and/or logic in the match determination process. For example, an application may wish to reduce a threshold depending on some application-specific data.

The full implementation of SNEvalNameData::getCompResult() appears in the SNEvalNameData.hpp header file. This gives the developer insight into how a customized method might incorporate additional data. The default method checks to see if the scores (gnScore, snScore, and nameScore) meet or exceed their respective thresholds. The thresholds are set via the SNQueryParms object associated with this evaluation name.

Parameters:

None.

Return Values:

An SNReturnCode value of either SN_MATCH or SN_NO_MATCH.

Examples:

The example below shows an override of the SNEvalNameData::getCompResult() method. Our example assumes a subclass of SNEvalNameData, then adds a Boolean ssnMatch flag that is set to true if the query and evaluation name SSNs match exactly. In our override, we reduce the thresholds by 10 percent when the ssnMatch flag is true.

```
SNReturnCode     MySNEvalNameData::getCompResult()
{
   SNReturnCode retCode;
   double           adjustedGnScoreThresh = queryParms->getGnScoreThresh();
   double           adjustedSnScoreThresh = queryParms->getSnScoreThresh();
   double           adjustedNameScoreThresh = queryParms->getNameScoreThresh

   if (ssnMatch)  {      // adjust the threshold if we have an exact ssn mat
      adjustedGnScoreThresh *= 0.9;
      adjustedSnScoreThresh *= 0.9;
      adjustedNameScoreThresh *= 0.9;
   }

   if ((nameScore >= adjustedNameScoreThresh) &&
       (gnScore >= adjustedGnScoreThresh) &&
       (snScore >= adjustedSnScoreThresh))
        retCode = SN_MATCH;
   else
        retCode = SN_NO_MATCH;

   return retCode;
}
```

`virtual void resetScores()`

Clears out the scores associated with the SNEvalNameData object. Developers that wish to reuse SNEvalNameData objects for multiple queries must call this function before each call to performComp(). If an application creates and deletes SNEvalNameData objects for each query it processes, this function is not necessary.

When subclassing SNEvalNameData, you should override this method to reset any scoring variables you have added. In doing so, be sure to call the base class's implementation.

Parameters:

None.

Return Values:

None.

Examples:

The example below shows the same SNEvalNameData object being used in two separate queries. Note the call to resetScores() before the second query is performed.

```
SNEvalNameData  *candidate;
SNQueryNameData *queryName1;
SNQueryNameData *queryName2;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
int             snSegDifferential;

candidate = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
queryName1 = new SNQueryNameData(queryParms, "James Earl", "Jones");

candidate->performComp(queryName);

candidate->resetScores();

queryName2 = new SNQueryNameData(queryParms, "Jimmy", "Jones");
candidate->performComp(queryName);

//      delete the allocated objects somewhere below.
```

SNAPI is a trademark of Language Analysis Systems. All other products mentioned are registered trademarks or trademarks of their respective companies.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Tuesday November 25, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNNameData Class

Overview

SNNameData encapsulates the basic information required to describe a name. It is the base class for both SNEvalNameData and SNQueryNameData.

The developer never instantiates an object of this class. However, SNNameData defines some members and member functions that are useful to applications, and they are documented here.

Subclassing

Applications should not subclass from SNNameData directly. Instead, subclasses should be derived from SNEvalNameData and SNQueryNameData as appropriate.

Methods Summary

Common Methods:

getGn() Returns the given name, in its original case.

getSn() Returns the surname, in its original case.

Attributes:

char gn[];

An NULL terminated array of characters that holds the original given name, in its original case. The array is large enough to hold SN_MAX_GN_LEN characters.

SNQueryParms *queryParms;

A pointer to the SNQueryParms object used to create this name object.

char gn[];

An NULL terminated array of characters that holds the original surname, in its original case. The array is large enough to hold SN_MAX_SN_LEN characters.

Method Details:

```
char * getGn();
```

Returns the given name, in its original case. This is primarily a convenience parameter, but can also be used to determine how the API separated a single name string into separate given name and surname fields.

Parameters:

None.

Return Values:

The given name as a NULL terminated string.

Examples:

The example below shows the construction of an SNEvalNameData object and a subsequent given name examination:

```
SNEvalNameData   *candidate;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);

candidate = new SNEvalNameData(queryParms, "Bob Earl Jones Jr",
                                SN_LAST_SEG_IS_SURNAME);

printf("The given name was %s\n", candidate->getGn();
      ...
//      delete the allocated objects somewhere below.
```

```
char * getSn();
```

Returns the surname, in its original case. This is primarily a convenience parameter, but can also be used to determine how the API separated a single name string into separate given name and surname fields.

Parameters:

None.

Return Values:

The given name as a NULL terminated string.

Examples:

```
The example below shows the construction of an SNEvalNameData object
and a subsequent surname examination:


SNEvalNameData  *candidate;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);

candidate = new SNEvalNameData(queryParms, "Bob Earl Jones Jr",
                               SN_LAST_SEG_IS_SURNAME);

printf("The surname was %s\n", candidate->getSn();

//      delete the allocated objects somewhere below.
```

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Tuesday November 25, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNQueryNameData Class

- Class Overview
- Subclassing
- Methods Summary
- Attributes
- Construction
- Method Details

Overview

A subclass of SNNameData, SNQueryNameData represents a query name that will be compared against many evaluation names (SNEvalNameData objects). Built on top of SNNameData, SNQueryNameData adds a mechanism to manage results (an SNResultsList object) through a series of comparisons. In addition, it adds variables to store pre-processing information about the query name, such as list of variant names.

The developer may subclass this class to add any data that might be useful to attach to a query name. Such additions are often done in tandem with similar additions to an analogous subclass of the SNEvalNameData class. See the subclassing section for more details.

The developer is responsible for deleting any SNQueryNameData objects created by their code. Typically, the developer will construct a single SNQueryNameData object, and compare it to multiple evaluation names (SNEvalNameData objects). Once the query is completed, and the results have been retrieved, the query object is deleted.

The developer may attach an SNResultsList Object to the SNQueryNameData object for the purpose of results management. The SNResultsList object handles issues of comparing and sorting evaluation names that are determined to be matches. In addition, the SNResultsList object can trim the set of matching names down to the best N names, where N is specified by the developer. Use of an SNResultsList object is optional - if desired, the developer can provide their own match management. See the SNResultsList documentation for a more detailed discussion.

Subclassing

A developer may wish to subclass SNQueryNameData to allow application-specific data to be incorporated into the search process. For example, suppose an application needs to search for names in a database that contains given name, surname, and birthdate. Suppose further that the application needs to include birthdate as a factor in the search. By subclassing both SNQueryNameData and SNEvalNameData, and adding a birthdate member data variable to each, the developer can override methods with SNEvalNameData (e.g. calcNameScore) to include age differential (how close in age the candidate is to the query) in the comparison.

The following are methods that can be overridden to provide specialized processing in a subclass:

~SNQueryNameData() Destructor for the class. Ensure that the destructor for your subclass frees any resources your subclass allocates.

Methods Summary

Common Methods:

SNQueryNameData() Various constructors for the class.

getResultsList() Returns the SNResultsList object associated with this query object.

setResultsList() Attaches an SNResultsList object to this query object.

Attributes:

SNResultsList *resultsList; A pointer to the SNResultsList object that is being used to manage the matches for the query object. If no results list has been attached, the value of this variable is NULL.

Method Details:

Constructors:

□	`SNQueryNameData(SNQueryParms *qParms, char *gn, char *sn);`
□	`SNQueryNameData(SNQueryParms *qParms, char *gn, char *sn, char *mn);`
□	`SNQueryNameData(SNQueryParms *qParms, char *name, SNNameFormat nameFormat);`

Each constructor creates a new SNQueryNameData object. All forms of the constructor take a pointer to an SNQueryParms object, which should be the same one used to create the SNEvalNameData object that this object will be compared against.

The SNAPI system is based internally on a name model that considers given name and surname. However, other constructors are provided for cases where an alternate format is desired. In these cases, the constructor maps the supplied data into SNAPI's given name/surname model.

The first form of the constructor takes a given name and surname. This is the most efficient and accurate form, because the data already corresponds to SNAPI's internal name model.

The second form accomodates systems that have knowledge of a middle name. Currently, SNAPI maps the middle name into the given name. Future versions of SNAPI may provide more sophisticated handling of middle name data.

The third form accomodates systems that represent names as a single string, rather than separate fields. This form takes an SNNameFormat parameter that dictates how the string will be mapped into the given name/surname model. Values for this parameter include:

SN_SURNAME_COMMA_GIVENNAME
Exects the name string in the form "surname, given name". If no comma is found in the string, the given name is assumed to be unknown, and the entire string is placed in the surname field.

SN_LAST_SEG_IS_SURNAME
Expects the string to be in the form "given_name surname". The last segment is placed in the surname field. All other segments are placed in the given name field. If a string has just one segment, that segment is placed in the surname field, and the given name field is assumed to be unknown. The processing is intelligent in that TAQ values are recognized when determining the last segment. This allows a name such as "Bob Jones Jr." to be mapped correctly with given name of "Bob" and a surname of "Jones Jr", rather than incorrectly assigning "Jr" as the surname.

SN_NAME_FORMAT_UNKNOWN
Currently operates identically to SN_LAST_SEG_IS_SURNAME. Future versions of SNAPI might use linguistic expertise to make automated decisions about parsing the string into name fields.

Parameters:

`qParms`
A pointer to an SNQueryParms object. This should be the same object used to create the SNEvalNameData objects that this SNQueryNameData object will be compared against.

`gn`
A NULL terminated string that represents the given name.

`sn`
A NULL terminated string that represents the surname.

`mn`
A NULL terminated string that represents the middle name.

`name`
A NULL terminated string that represents all components of the name as a single string.

`nameFormat`
An enumerated type value that specifies how to interpret the name string when breaking it into given name and surname. See documentation for SNNameFormat for valid values.

Return Values:

None.

Memory Management:

The responsibility for deleting an SNQueryNameData lies with the developer. In general, an SNQueryNameData object should be deleted shortly after it has been compared to all SNEvalNameData objects that need to be considered for the query, and after all results from the query have been retrieved.

Examples:

```
The example below shows four equivalent SNQueryNameData objects being
constructed. In each case, the supplied name gets mapped to an
internal representation of: Given Name = "Bob Earl", Surname =
"Jones".


SNQueryNameData *query1;
SNQueryNameData *query2;
SNQueryNameData *query3;
SNQueryNameData *query4;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);

query1 = new SNQueryNameData(queryParms, "Bob Earl", "Jones");
query2 = new SNQueryNameData(queryParms, "Bob", "Jones", "Earl");
query3 = new SNQueryNameData(queryParms, "Bob Earl Jones",
                                   SN_LAST_SEG_IS_SURNAME);
query4 = new SNQueryNameData(queryParms, "Jones, Bob Earl",
                                   SN_SURNAME_COMMA_GIVENNAME);

// delete the allocated objects somewhere below.
```

SNResultsList * getResultsList()

Returns the SNResultsList object associated with this query object. If no SNResultsList object has been associated with this query object, the function returns NULL. In general, the developer does not need to call this function because they already have a pointer to the results list object (since they created it).

Parameters:

None.

Return Values:

A pointer to the SNResultsList object associated with this query object. If no SNResultsList object has been associated with this query object, the function returns NULL.

`void setResultsList(SNResultsList *aResultsList)`

Sets the resultsList member variable. Call this member function to attach an SNResultsList object to the query object. In general, an application will create a new SNResultsList object for each query, and pass a pointer to the SNResulstList object to setResultsList(). After the query is completed, the SNResultsList object should be deleted.

Parameters:

`aResultsList` A pointer to a SNResultsList object that will manager the matches for this query.

Return Values:

None.

Examples:

```
The example below shows a sample query session using an SNResultsList
object:


SNEvalNameData  *candidate1;
SNEvalNameData  *candidate2;
SNQueryNameData *queryName;
SNQueryParms    *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode    retCode;
SNResultsList   *myResultsList = NULL;

candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
candidate2 = new SNEvalNameData(queryParms, "Earl", "Jhonas");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

myResultsList = new SNResultsList(1);   // create a manager for just 1 matc
queryName->setResultsList(myResultsList);

candidate1->performComp(queryName);
candidate2->performComp(queryName);

delete candidate1;
delete candidate2;

if (myResultsList->getNumHits() > 0)    {
   SNEvalNameData *matchName = myResultsList->getHitAt(0);
   printf("best match was %s, %s\n", matchName->getSn(), matchName->getGn()
}
else
   printf("Neither name Matched");

delete myResultsList;
delete queryName;
```

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Tuesday November 25, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNQueryParms Class

Overview

The SNQueryParms class encapsulates all the tunable parameters that determine how a name is processed and compared to another name. A simple application can create a single SNQueryParms object, adjust it accordingly, and use it to perform all query processing. A more complex application might need to re-adjust parameter settings for each query, perhaps based on user selections.

An SNQueryParms object provides access to over forty parameters. Many of these parameters provide highly specialized tuning required only for particular circumstances. Other specialized parameters are used to address the nuances of names within a certain culture.

In an effort to shield the developer from the complexity of these numerous parameters, the API provides sets of pre-defined default parameters. These sets are organized by culture - for example the Hispanic parameters set contains values suitable for evaluating Hispanic names. In general, most applications need to adjust only a few of the available parameters to achieve desired results. The availablecultural parameter packages and their default valuesfor each parameter are available for inspection. The developer creates a set of parameters by constructing an SNQueryParms object, which takes a cultural specifier as an argument. The culture used to create a parameters object also determines the subset ofTAQ values and variants that will be used when processing names created with the parameters object.

Parameters often specify factors, thresholds, or scores. It is important to understand the distinction between each of these:

Factor A factor is a number that is applied to an existing score to arrive at a new sco For example, when comparing two name segments that are out of place, the

OOPS Factor is applied to the segment score to arrive at a new (lower) score.

Threshold A threshold is a score that a comparison must achieve in order to be consider match. SNQueryParms defines thresholds for the given name score, the surn score, and the composite name score (which considers the name as a whole).

Score A score specifies a value to assign in a particular situation. For example, SNQueryParms defines a *first name unknown* score, which specifies the scor assign to a segment comparison when one of the segments is unknown. Note a factor can still be applied to a score after it has been assigned.

The documentation below organizes the methods of this class into usage categories of *Common*, *Specialized*, and *Advanced*.

- *Common* methods adjust the most basic parameters, involving minimal complexity. Most applications will only need to use methods from this category.
- *Specialized* methods are slightly more complex in nature, requiring a basic understanding of the name scoring process to be used properly.
- *Advanced* methods address very specific behaviors within the name comparison, and require a deep understanding of the issues involved in name analysis. Because the API provides default values for these settings, most applications will never need to use these methods.

Because SNQueryParms provides methods to retrieve and set the value of each parameter, both methods are presented together. Further, many parameters operate on a particular name field, and therefore exist in pairs (one that affects given name processing, and another that affects surname processing). Because these pairs of functions are in all other respects identical, full documentation is provided with the methods that operate on the given name. The analogous function for the surname references the detail presented for the given name function.

Subclassing

In general, an application should not need to subclass SNQueryParms. However, an advanced application may need to introduce a new, customized parameter that will be referenced during name comparisons.

For example, an application that uses Social Security Number when comparing names might introduce a tunable parameter called ssnScoreTheshold. This parameter would specify an SSN score that a name would need to beat in order to be considered a match. The parameter value would then be compared to the ssnScore in the application's override of the SNEvalNameData:getCompResult() method. See the SNEvalNameData class and its calcComponentScores() method for more details.

Note that in the above example, subclassing SNQueryParms is necessary only if we need a tunable threshold parameter. If, on the other hand, the value of the threshold is a fixed value, the threshold value itself can be specified in the override of the SNEvalNameData::getCompResult() method.

The following are methods that can be overridden to provide specialized processing in a subclass:

~SNQueryParms() Destructor for the class. Ensure that the destructor for your subclass any resources your subclass allocates.

loadFromFile() Loads in parameter values from a file.

saveToFile() Writes out parameter values to a file.

Methods Summary

Common Methods:

SNQueryParms() Constructor for the class.

getGnScoreThresh()
setGnScoreThresh() Gets or sets the given name score threshold (the lowest given name score a name can receive and still be considered a match).

getGnWeight()
setGnWeight() Gets or sets the given name weight. This parameter controls the importance of the given name score (relative to the surname score) when computing the composite name score.

getScoreThresh()
setScoreThresh() Gets or sets the overall name score threshold (the lowest score a name can receive and still be considered a match).

getSnScoreThresh()
setSnScoreThresh() Gets or sets the surname score threshold (the lowest surname score a name can receive and still be considered a match).

getSnWeight()
setSnWeight() Gets or sets the surname weight. This parameter controls the importance of the surname score (relative to the give name score) when computing the composite name score.

getStatus() Returns the current status of the object. This function is used to ensure the successful construction of the object.

Specialized Methods:

getCheckGnUnknowns()
setCheckGnUnknowns()

Gets or sets the flag that determines if given name segments should be checked for the special strings "NFN", "NMN", "FNU", and "MNU" when processing names. These strings are commonly used in legacy systems to indicate that a name is unknown or does not exist.

getCheckSnUnknowns()
setCheckSnUnknowns()

Gets or sets the flag that determines if surname segments should be checked for the special strings "NLN" and "LNU" when processing names. These strings are commonly used in legacy systems to indicate that a name is unknown or does not exist.

getFNUScore()
setFNUScore()

Gets or sets the score to assign to a given name segment comparison where the given name is unknown. A given name segment is considered unknown if it is blank, or if it is specified as "FNU" or "MNU".

getGnInitialOnInitialMatchScore()
setGnInitialOnInitialMatchScore()

Gets or sets the score to assign to a given name segment comparison where both segments are initials (assuming given name initial matching is turned on via setMatchGnInitial()).

getGnInitialScore()
setGnInitialScore()

Gets or sets the score to assign to a given name segment comparison involving an initial (assuming given name initial matching is turned on via setMatchGnInitial()).

getGnSegmentScoreMode()
setGnSegmentScoreMode()

Gets or sets the given name segment score mode. The parameter determines how to handle multi-segment names.

getLNUScore()
setLNUScore()

Gets or sets the score to assign to a surname segment comparison where the surname is unknown. A surname segment is considered unknown if it is blank, or if it is specified as "LNU".

getMatchGnIntial()
setMatchGnIntial()

Gets or sets the flag that determines if a given name segment comparison should give special consideration to initials.

getMatchSnIntial()
setMatchSnIntial()

Gets or sets the flag that determines if a surname segment comparison should give special consideration to initials.

getNFNScore()
setNFNScore()

Gets or sets the score to assign to a given name segment comparison where the given name does not exist. Note that a blank name is considered to be unknown. Only given name segments specified as "NFN" or "NMN" are considered non-existent.

getNLNScore()
setNLNScore()

Gets or sets the score to assign to a surname segment comparison where the surname does not exist. Note that a blank name is considered to be unknown. Only given name segments specified as "NLN" are considered non-existent.

getNoiseChars()
setNoiseChars()

Gets or sets the set of characters that are discarded when processing a name.

getSegmentBreakChars()
setSegmentBreakChars()

Gets or sets the set of characters that are considered segment separators.

getSnInitialScore()
setSnInitialScore()

Gets or sets the score to assign to a surname segment comparison where one segment is an initial (assuming surname initial matching is turned on via setMatchSnInitial()).

getSnInitialOnInitialMatchScore()
setSnInitialOnInitialMatchScore()

Gets or sets the score to assign to a surname segment comparison where both segments are initials (assuming surname initial matching is turned on via setMatchSnInitial()).

getSnSegmentScoreMode()
setSnSegmentScoreMode()

Gets or sets the surname segment score mode. The parameter determines how to handle multi-segment names.

Advanced Methods:

getAbsDelGnTAQFactor()
setAbsDelGnTAQFactor()

Gets or sets the factor to apply to a given name segment score when a delete TAQ value is associated with one segment, but no delete TAQ value is associated with the other segment. The factor is applied only if given nameTAQ scoring is enabled (see setGnTAQProcessingMode()).

getAbsDelSnTAQFactor()
setAbsDelSnTAQFactor()

Gets or sets the factor to apply to a surname segment score when a delete TAQ value is associated with one segment, but no delete TAQ value is associated with the other segment. The factor is applied only if surname TAQ scoring is enabled (see setSnTAQProcessingMode()).

getAbsDisGnTAQFactor()
setAbsDisGnTAQFactor()

Gets or sets the factor to apply to a given name segment score when a disregard TAQ value is associated with one segment, but no disregard TAQ value is associated with the other segment. The factor is applied only if given name TAQ scoring is enabled (see setGnTAQProcessingMode()).

getAbsDisSnTAQFactor()
setAbsDisSnTAQFactor()

Gets or sets the factor to apply to a surname segment score when a disregard TAQ value is associated with one segment, but no disregard TAQ value is associated with the other segment. The factor is applied only if surname TAQ scoring is enabled (see setSnTAQProcessingMode()).

getCheckGnCompressedName()
setCheckGnCompressedName()
	Gets or sets the flag that determines if a compressed name comparison should be performed on the given name. See the method details for a description of the compressed name check.

getCheckSnCompressedName()
setCheckSnCompressedName()
	Gets or sets the flag that determines if a compressed name comparison should be performed on the surname. See the method details for a description of the compressed name check.

getDelGnTAQFactor()
setDelGnTAQFactor()
	Gets or sets the factor to apply to a given name segment score when a delete TAQ value is associated with one segment, and a different delete TAQ value is associated with the other segment. The factor is applied only if given name TAQ scoring is enabled (see setGnTAQProcessingMode()).

getDelSnTAQFactor()
setDelSnTAQFactor()
	Gets or sets the factor to apply to a surname segment score when a delete TAQ value is associated with one segment, and a different delete TAQ value is associated with the other segment. The factor is applied only if surname TAQ scoring is enabled (see setSnTAQProcessingMode()).

getDisGnTAQFactor()
setDisGnTAQFactor()
	Gets or sets the factor to apply to a given name segment score when a disregard TAQ value is associated with one segment, and a different disregard TAQ value is associated with the other segment. The factor is applied only if given name TAQ scoring is enabled (see setGnTAQProcessingMode()).

getDisSnTAQFactor()
setDisSnTAQFactor()
	Gets or sets the factor to apply to a surname segment score when a disregard TAQ value is associated with one segment, and a different disregard TAQ value is associated with the other segment. The factor is applied only if surname TAQ scoring is enabled (see setSnTAQProcessingMode()).

getGnAnchorFactor()
setGnAnchorFactor()
	Gets or sets the factor to apply to a given name segment score when the two segments are in place, but their ordinal position is not the anchor segment (as specified with the setGnAnchorSegmentMode() method).

getGnAnchorSegmentMode()
setGnAnchorSegmentMode()
	Gets or sets the given name anchor segment as either first, last, or none.

getGnCompressedNameScore()
setGnCompressedNameScore()
	Gets or sets the score assigned when two given names match via the compressed name check.

getGnOOPSFactor()
setGnOOPSFactor()
	Gets or sets the factor to apply to a given name segment score when the two segments are out of place (their ordinal position within the name field is different). Note that the anchor segment setting affects the determination of a

segment's ordinal position.

getGnTAQProcessingMode()
setGnTAQProcessingMode()
Gets or sets the TAQ processing mode for the given name field.

getSnAnchorFactor()
setSnAnchorFactor()
Gets or sets the factor to apply to a surname segment score when the two segments are in place, but their ordinal position is not the anchor segment (as specified with the setSnAnchorSegmentMode() method).

getSnAnchorSegmentMode()
setSnAnchorSegmentMode()
Gets or sets the surname anchor segment as either first, last, or none.

getSnCompressedNameScore()
setSnCompressedNameScore()
Gets or sets the score assigned when two surnames match via the compressed name check.

getSnOOPSFactor()
setSnOOPSFactor()
Gets or sets the factor to apply to a surname segment score when the two segments are out of place (their ordinal position within the name field is different). Note that the anchor segment setting affects the determination of a segment's ordinal position.

getSnTAQProcessingMode()
setSnTAQProcessingMode()
Gets or sets the TAQ processing mode for the surname field.

getUseGnLeftBias()
setUseGnLeftBias()
Gets or sets the flag that determines if character based given name segment comparisons will place more emphasis on leading characters.

getUseGnVariants()
setUseGnVariants()
Gets or sets the flag that determines if the API will reference its internal list of variants when processing given name segments.

getUseSnLeftBias()
setUseSnLeftBias()
Gets or sets the flag that determines if character based surname segment comparisons will place more emphasis on leading characters.

getUseSnVariants()
setUseSnVariants()
Gets or sets the flag that determines if the API will reference its internal list of variants when processing surname segments.

Attributes:

All attributes of the SNQueryParms class are protected and should not be accessed directly. Use the get and set methods for the desired attribute to inspect or set a particular attribute.

getSegmentBreakChars() returns a pointer to the API's copy of current segment break characters.

```
double          getSnInitialScore ()
SNReturnCode    setSnInitialScore(double aScore)
```

Gets or sets the surname "Initial Match" score. This is the score to assign during a segment comparison when one or the other segment (but not both) is an initial (it consists of just one character). In order for the score to be assigned, surname initial matching must have been turned on via thesetMatchSnIntial()method. Otherwise, the segments are compared via the standard character string comparison. See the setSnInitialOnInitialMatchScore()function for detail on how the API handles comparisons where both segments are initials.

Parameters:

`aScore` A double value between 0.0 and 1.0 inclusive. Any value outside this range an error.

Return Values:

setSnInitialScore() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_SN_INIT_SCORE	The specified score is invalid.

getSnInitialScore() returns the current "Initial Match" score (a double).

```
double          getSnInitialOnInitialMatchScore ()
SNReturnCode    setSnInitialOnInitialMatchScore(double aScore)
```

Gets or sets the surname "Initial on Initial Match" score. This is the score to assign during a segment comparison when both segments are initials (they consist of just one character). In order for the score to be assigned, surname initial matching must have been turned on via the setMatchSnIntial()method. Otherwise, the segments are compared via the standard character string comparison.

This method is provided to give applications more control over how initials are treated during a comparison. Most systems consider an initial match to be somewhat less exact than an exact match. For example, the surnames "Jones" and "J" do not match as closely as "Jones" and "Jones". However, the score to assign to the given names "R" and "R" is subject to interpretation by the application. Such a comparison could be considered an initial match, an exact match, or something in between. By providing this funable parameter, the API give the developer the ability to decide exactly how such situations should be handled.

Parameters:

aScore	A double value between 0.0 and 1.0 inclusive. Any value outside this range an error.

Return Values:

setSnInitialOnInitialMatchScore() returns arSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was suc
SN_INVALID_SN_INIT_ON_INIT_MATCH_SCORE	The specified score is inva

getSnInitialOnInitialMatchScore() returns the current surname "Initial On Initial Match" score (a double).

```
SNSegScoreMode    getSnSegmentScoreMode ()
void              setSnSegmentScoreMode(SNSegScoreMode aMode)
```

Gets or sets the surname segment score mode.

The surname segment score mode governs how the API computes a surname score when the both surnames involved in the comparison have more than one segment. See the analogous setGnSegmentScoreMode() method for details.

Parameters:

aMode	An SNSegScore value of SN_SEGMODE_HIGHEST, SN_SEGMODE_AV or SN_SEGMODE_LOWEST.

Return Values:

getSnSegmentScoreMode() returns the current surname segment score mode.

```
double            getAbsDelGnTAQFactor ()
SNReturnCode      setAbsDelGnTAQFactor(double aFactor)
```

Gets or sets the given name "absent delete TAQ" factor. The "absent delete TAQ" factor is applied to a segment score when one of the segments has an associated delete TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDelGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DEL_GN_TAQ_FACTOR	The specified factor is invalid.

getAbsDelGnTAQFactor() returns the current "absent delete TAQ" factor.

```
double            getAbsDelSnTAQFactor ()
SNReturnCode      setAbsDelSnTAQFactor(double aFactor)
```

Gets or sets the surname "absent delete TAQ" factor. The "absent delete TAQ" factor is applied to a segment score when one of the segments has an associated delete TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion onTAQs for an explanation of the different types of TAQ values. See the discussion onTAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDelSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DEL_SN_TAQ_FACTOR	The specified factor is invalid.

getAbsDelSnTAQFactor() returns the current "absent delete TAQ" factor.

```
double          getAbsDisGnTAQFactor ()
SNReturnCode    setAbsDisGnTAQFactor(double aFactor)
```

Gets or sets the given name "absent disregard TAQ" factor. The "absent disregard TAQ" factor is applied to a segment score when one of the segments has an associated disregard TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDisGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DIS_GN_TAQ_FACTOR	The specified factor is invalid.

getAbsDisGnTAQFactor() returns the current "absent disregard TAQ" factor.

```
double          getAbsDisSnTAQFactor ()
SNReturnCode    setAbsDisSnTAQFactor(double aFactor)
```

Gets or sets the surname "absent disregard TAQ" factor. The "absent disregard TAQ" factor is applied to a segment score when one of the segments has an associated disregard TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDisSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DIS_SN_TAQ_FACTOR	The specified factor is invalid.

getAbsDisSnTAQFactor() returns the current "absent disregard TAQ" factor.

```
BOOL      getCheckGnCompressedName ()
void      setCheckGnCompressedName(BOOL aBool)
```

Gets or sets the flag that determines if a compressed name comparison should be performed on the given name.

After the given name has been score, the API can optionally perform a compressed name comparison on the given name. For this comparison, all segment break characters and noise characters are removed from both the query and evaluation given names. If the two strings match exactly, the given name score is set to the given name compressed name score (setGnCompressedNameScore()), unless the existing given name score is already higher than the given name compressed name score.

The given name compressed name check can be though of as a way to squeeze all of a given name's segments together. This can help solve problems associated with discrepancies in the segmentation of names.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool`	A BOOL value of TRUE or FALSE.

Return Values:

getCheckGnCompressedName() returns the current value of the flag.

```
BOOL      getCheckSnCompressedName ()
void      setCheckSnCompressedName(BOOL aBool)
```

Gets or sets the flag that determines if a compressed name comparison should be performed on the surname.

After the surname has been scored, the API can optionally perform a compressed name comparison on the surname. For this comparison, all segment break characters and noise characters are removed from both the query and evaluation given names. If the two strings match exactly, the surname score is set to the surname compressed name score (setSnCompressedNameScore()), unless the existing surname score is already higher than the surname compressed name score.

The surname compressed name check can be thought of as a way to squeeze all of a surname's segments together. This can help solve problems associated with discrepancies in the segmentation of names.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getCheckSnCompressedName() returns the current value of the flag.

intentionally blank

intentionally blank

TAQs, but no disregard TAQ value is common to both segments. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setDisGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_DIS_GN_TAQ_FACTOR	The specified factor is invalid.

getDisGnTAQFactor() returns the current "disregard TAQ" factor.

```
double            getDisSnTAQFactor ()
SNReturnCode      setDisSnTAQFactor(double aFactor)
```

Gets or sets the surname "disregard TAQ" factor. The "disregard TAQ" factor is applied to a segment score when both segments have one or more associated disregard TAQs, but no disregard TAQ value is common to both segments. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods ~~and~~ should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setDisSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_DIS_SN_TAQ_FACTOR	The specified factor is invalid.

getDisSnTAQFactor() returns the current "disregard TAQ" factor.

```
double           getGnAnchorFactor ()
SNReturnCode     setGnAnchorFactor(double aFactor)
```

Gets or sets the factor to apply to a given name segment score when the two segments are in place, but their ordinal position is not the anchor segment (as specified with the setGnAnchorSegmentMode() method).

The anchor factor should be viewed as a way to diminish the importance of a match if the match occurs between two segments that are not in the anchor segment position. For example, Arabic given names commonly include one or more segments. The first segment is the more stable segment and should therefore be considered the anchor segment. A match between two segments in the second given name position is considered to be of less importance (relative to the first segment), and as such, that segment score is diminished by applying the anchor factor.

Note that the given name anchor factor is only applied when the two segments are in place (they are in the same position). Given name segments that are out of place are adjusted by the given name "out of place segment" score (setGnOOPSFactor()). In addition, the given name anchor factor is only applied when the given name anchor segment mode (setGnAnchorSegmentMode()) has been set.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnAnchorFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_GN_ANCHOR_FACTOR	The specified factor is invalid.

getGnAnchorFactor() returns the current "given name anchor segment" factor.

```
SNAnchorSegMode    getGnAnchorSegmentMode ()
void               setGnAnchorSegmentMode(SNAnchorSegMode anAnchorMode)
```

Gets or sets the given name anchor segment mode. Setting the anchor segment mode causes the API to place emphasis on a particular segment within the given name (the first segment, or the last segment). When this feature is turned off, all segments are considered to be equally important. See thesetGnAnchorFactor() method for details on how the anchor segment affects segment scoring.

The given name anchor segment is also used to determine how segments in two names are lined up (to determine which segments are in place or out of place). When the anchor segment is set to SN_ANCHOR_SEG_NONE or SN_ANCHOR_SEG_FIRST, segment alignment starts from the left (the first segment). When the anchor segment is set to SN_ANCHOR_SEG_LAST, segment alignment starts from the right (the last segment). See thesetGnOOPSFactor() method for details on how the API adjusts the score of segments that are out of place.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`anAnchorMode`	A SNAnchorSegMode value:	
	SN_ANCHOR_SEG_NONE	No segment carries more importanc another. Name segments are lined u left to determine which segment co are in place.
	SN_ANCHOR_SEG_FIRST	The first segment is the most impor segment. Name segments are lined left to determine which segment co are in place.
	SN_ANCHOR_SEG_LAST	The last (right most) is the most im segment. Name segments are lined right to determine which segment comparisons are in place.

Return Values:

getGnAnchorSegmentMode() returns the current "given name anchor segment" mode.

```
double          getGnCompressedNameScore ()
SNReturnCode    setGnCompressedNameScore(double aScore)
```

Gets or sets the score to assign to a successfulgiven name compressed name comparison. See the setCheckGnCompressedName() method for detail on compressed-name comparisons.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aScore` A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnCompressedNameScore() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was succ
SN_INVALID_GN_COMPRESSED_NAME_SCORE	The specified score is inval

getGnCompressedNameScore() returns the current "given name compressed name" score.

```
double          getGnOOPSFactor ()
SNReturnCode    setGnOOPSFactor(double aFactor)
```

Gets or sets thegiven name "out of place segment" factor. This is the factor that is applied to a segment score when the two segments are out of place (their ordinal positions are different).The given name anchor segment mode (setGnAnchorSegMode()) affects how segment alignment is performed.

To understand how alignment affects in place/out of place determination, consider the given names "Earl Bob" and "James Earl Bob". If we align these names on the left, we get:

Name 1:	Earl	Bob	
Name 2:	James	Earl	Bob

If we line the names up on the right, we get:

Name 1:		Earl	Bob
Name 2:	James	Earl	Bob

Notice that in the first case, the "Earl" and "Bob" segments are out of place, so we would apply the given name "out of place segment" factor to their segment scores. In the second case, because we align on the right, the segments are in place, so their segment scores are not adjusted.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnOOPSFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_GN_OOPS_FACTOR	The specified factor is invalid.

getGnOOPSFactor() returns the current given name "out of place segment" factor.

```
SNTAQProcessingMode    getGnTAQProcessingMode ()
void                   setGnTAQProcessingMode(SNTAQProcessingMode aMode)
```

Gets or sets the mode that determines how to processgiven name TAQ values.

The following modes are supported:.

Mode	Description
SN_TAQ_MODE_IGNORE	The API will not check given name segments t are TAQ values.
SN_TAQ_MODE_JUST_REMOVE	The API will check each given name segment t a TAQ value. If so, the value is removed as tho existed.
SN_TAQ_MODE_IGNORE	The API will check each given name segment t a TAQ value. If so, the segment gets associated proper stem segment, and is used in the compu stem segment's score.

See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aMode	An SNTAQProcessingMode value of either SN_TAQ_MODE_IGNORE or SN_TAQ_MODE_JUST_REMOVE.

Return Values:

getGnTAQProcessingMode() returns the current given name TAQ processing mode.

```
double            getSnAnchorFactor ().
SNReturnCode      setSnAnchorFactor(double aFactor)
```

Gets or sets the factor to apply to asurname segment score when the two segments are in place, but their ordinal position is not the anchor segment (as specified with the setSnAnchorSegmentMode()method).

The anchor factor should be viewed as a way to diminish the importance of a match if the match occurs between two segments that are not in the anchor segment position. For example, Hispanic surnames commonly include two segments. The first segment is the true surname and should therefore be considered the anchor segment. A match between two segments in the second position is considered to be of less importance (relative to the first segment), and as such, that segment score is diminished by applying the anchor factor.

Note that the surname anchor factor is only applied when the two segments are in place (they are in the same position). Surname segments that are out of place are adjusted by the surname "out of place segment" score (getSnOOPSFactor()). In addition, the surname anchor factor is only applied when the surname anchor segment mode (setSnAnchorSegmentMode()) has been set.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnAnchorFactor() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_SN_ANCHOR_FACTOR	The specified factor is invalid.

getSnAnchorFactor() returns the current "surname anchor segment" factor.

```
SNAnchorSegMode    getSnAnchorSegmentMode ()
void               setSnAnchorSegmentMode(SNAnchorSegMode anAnchorMode)
```

Gets or sets the surname anchor segment mode. Setting the anchor segment mode causes the API to place emphasis on a particular segment within the surname (the first segment, or the last segment). When this feature is turned off, all segments are considered to be equally important. See thesetSnAnchorFactor() method for details on how the anchor segment affects segment scoring.

The surname anchor segment is also used to determine how segments in two names are lined up (to determine which segments are in place or out of place). When the anchor segment is set to SN_ANCHOR_SEG_NONE or SN_ANCHOR_SEG_FIRST, segment alignment starts from the left (the first segment). When the anchor segment is set to SN_ANCHOR_SEG_LAST, segment alignment starts from the right (the last segment). See thesetSnOOPSFactor() method for details on how the API adjusts the score of segments that are out of place.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`anAnchorMode` A SNAnchorSegMode value:

SN_ANCHOR_SEG_NONE	No segment carries more importance than another. Name segments are line up on the left to determine which segment comparisons are in place.
SN_ANCHOR_SEG_FIRST	The first segment is the most importan segment. Name segments are lined up. the left to determine which segment comparisons are in place.

SN_ANCHOR_SEG_LAST	The last (right most) is the most important segment. Name segments ar lined up on the right to determine whi segment comparisons are in place.

Return Values:

getSnAnchorSegmentMode() returns the current "surname anchor segment" mode.

```
double          getSnCompressedNameScore ()
SNReturnCode    setSnCompressedNameScore(double aScore)
```

Gets or sets the score to assign to a successfulsurname compressed name comparison. See the setCheckSnCompressedName() method for detail on compressed name comparisons.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aScore`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnCompressedNameScore() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was succe

SN_INVALID_SN_COMPRESSED_NAME_SCORE The specified score is inval

getSnCompressedNameScore() returns the current "surname compressed name" score.

```
double          getSnOOPSFactor ()
SNReturnCode    setSnOOPSFactor(double aFactor)
```

Gets or sets the surname "out of place segment" factor. This is the factor that is applied to a segment score when the two segments are out of place (their ordinal positions are different).The surname anchor segment mode (getSnAnchorSegMode()) affects how segment alignment is performed.

To understand how alignment affects in place/out of place determination, consider the surnames "Garcia Gomez " and "Valdez Garcia Gomez". If we align these names on the left, we get:

Name 1:	Garcia	Gomez	
Name 2:	Valdez	Garcia	Gomez

If we line the names up on the right, we get:

Name 1:		Garcia	Gomez
Name 2:	Valdez	Garcia	Gomez

Notice that in the first case, the "Garcia" and "Gomez" segments are out of place, so we would apply the surname "out of place segment" factor to their segment scores. In the second case, because we align on the right, the segments are in place, so their segment scores are not adjusted.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnOOPSFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_SN_OOPS_FACTOR	The specified factor is invalid.

getSnOOPSFactor() returns the current surname "out of place segment" factor.

```
SNTAQProcessingMode    getSnTAQProcessingMode ()
void                   setSnTAQProcessingMode(SNTAQProcessingMode aMode)
```

Gets or sets the mode that determines how to processsurname TAQ values.

The following modes are supported:

Mode	**Description**
SN_TAQ_MODE_IGNORE	The API will not check surname segments to s TAQ values.
SN_TAQ_MODE_JUST_REMOVE	The API will check each surname segment to s TAQ value. If so, the value is removed as thou existed.
SN_TAQ_MODE_IGNORE	The API will check each surname segment to s TAQ value. If so, the segment gets associated proper stem segment, and is used in the compu stem segment's score.

See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aMode An SNTAQProcessingMode value of either SN_TAQ_MODE_IGNORE or SN_TAQ_MODE_JUST_REMOVE.

Return Values:

getSnTAQProcessingMode() returns the current surname TAQ processing mode.

```
BOOL      getUseGnLeftBias()
void      setUseGnLeftBias(BOOL aBool)
```

Gets or sets the flag that determines if given name segment comparisons should be biased towards matches that occur at the beginning of the segment. When this feature is turned on, as we move to the right, matching character pairs are given decreasingly less credit in calculating a segment score. When this feature is turned off, all matching character pairs receive full credit, regardless of their position with their respective segment.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aBool A BOOL value of TRUE or FALSE.

Return Values:

getUseGnLeftBias() returns the current value of the flag (TRUE or FALSE).

```
BOOL     getUseGnVariants()
void     setUseGnVariants(BOOL aBool)
```

Gets or sets the flag that determines if given name segment comparisons should check to see if the two segments are linguistic variants of each other.

The API maintains internal tables that describe relationships between name variants. Each variant relationship has an associated score and culture. When comparing two segments, the API examines the value of the "use given name variants" flag. If it is turned on, the internal variant tables are searched to see if there is a variant relationship between the two segments, within the culture associated with this query (as determined by the SNQueryParms object used to perform the comparison). There is also a generic set of variants that are searched independent of culture. If a variant relationship is found, its associated score is assigned to the segment score, and no character based comparison is performed.

At present, the set of variants and their associated scores can not be modified by the developer.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getUseGnVariants() returns the current value of the flag (TRUE or FALSE).

```
BOOL      getUseSnLeftBias()
void      setUseSnLeftBias(BOOL aBool)
```

Gets or sets the flag that determines ifsurname segment comparisons should be biased towards matches that occur at the beginning of the segment. When this feature is turned on, as we move to the right, matching character pairs are given decreasingly less credit in calculating a segment score. When this feature is turned off, all matching character pairs receive full credit, regardless of their position with their respective segment.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getUseSnLeftBias() returns the current value of the flag (TRUE or FALSE).

```
BOOL     getUseSnVariants()
void     setUseSnVariants(BOOL aBool)
```

```
SNSegScoreMode     getSnSegmentScoreMode ()
void               setSnSegmentScoreMode(SNSegScoreMode aMode)
```

Gets or sets the surname segment score mode.

The surname segment score mode governs how the API computes a surname score when the both surnames involved in the comparison have more than one segment. See the analogous setGnSegmentScoreMode() method for details.

Parameters:

aMode	An SNSegScore value of SN_SEGMODE_HIGHEST, SN_SEGMODE_AV or SN_SEGMODE_LOWEST.

Return Values:

getSnSegmentScoreMode() returns the current surname segment score mode.

```
double             getAbsDelGnTAQFactor ()
SNReturnCode       setAbsDelGnTAQFactor(double aFactor)
```

Gets or sets the given name "absent delete TAQ" factor. The "absent delete TAQ" factor is applied to a segment score when one of the segments has an associated delete TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aFactor A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDelGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DEL_GN_TAQ_FACTOR	The specified factor is invalid.

getAbsDelGnTAQFactor() returns the current "absent delete TAQ" factor.

```
double          getAbsDelSnTAQFactor ()
SNReturnCode    setAbsDelSnTAQFactor(double aFactor)
```

Gets or sets the surname "absent delete TAQ" factor. The "absent delete TAQ" factor is applied to a segment score when one of the segments has an associated delete TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDelSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DEL_SN_TAQ_FACTOR	The specified factor is invalid.

getAbsDelSnTAQFactor() returns the current "absent delete TAQ" factor.

```
double          getAbsDisGnTAQFactor ()
SNReturnCode    setAbsDisGnTAQFactor(double aFactor)
```

Gets or sets the given name "absent disregard TAQ" factor. The "absent disregard TAQ" factor is applied to a segment score when one of the segments has an associated disregard TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDisGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DIS_GN_TAQ_FACTOR	The specified factor is invalid.

getAbsDisGnTAQFactor() returns the current "absent disregard TAQ" factor.

```
double            getAbsDisSnTAQFactor ()
SNReturnCode      setAbsDisSnTAQFactor(double aFactor)
```

Gets or sets the surname "absent disregard TAQ" factor. The "absent disregard TAQ" factor is applied to a segment score when one of the segments has an associated disregard TAQ, but the other does not. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setAbsDisSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_ABS_DIS_SN_TAQ_FACTOR	The specified factor is invalid.

getAbsDisSnTAQFactor() returns the current "absent disregard TAQ" factor.

```
BOOL      getCheckGnCompressedName ().
void      setCheckGnCompressedName(BOOL aBool)
```

Gets or sets the flag that determines if a compressed name comparison should be performed on the given name.

After the given name has been score, the API can optionally perform a compressed name comparison on the given name. For this comparison, all segment break characters and noise characters are removed from both the query and evaluation given names. If the two strings match exactly, the given name score is set to the given name compressed name score (setGnCompressedNameScore()), unless the existing given name score is already higher than the given name compressed name score.

The given name compressed name check can be though of as a way to squeeze all of a given name's segments together. This can help solve problems associated with discrepancies in the segmentation of names.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aBool	A BOOL value of TRUE or FALSE.

Return Values:

getCheckGnCompressedName() returns the current value of the flag.

```
BOOL      getCheckSnCompressedName ()
void      setCheckSnCompressedName(BOOL aBool)
```

Gets or sets the flag that determines if a compressed name comparison should be performed on the surname.

After the surname has been scored, the API can optionally perform a compressed name comparison on the surname. For this comparison, all segment break characters and noise characters are removed from both the query and evaluation given names. If the two strings match exactly, the surname score is set to the surname compressed name score (setSnCompressedNameScore()), unless the existing surname score is already higher than the surname compressed name score.

The surname compressed name check can be thought of as a way to squeeze all of a surname's segments together. This can help solve problems associated with discrepancies in the segmentation of names.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getCheckSnCompressedName() returns the current value of the flag.

intentionally blank

intentionally blank

TAQs, but no disregard TAQ value is common to both segments. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor` A double value between 0.0 and 1.0 inclusive.

Return Values:

setDisGnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_DIS_GN_TAQ_FACTOR	The specified factor is invalid.

getDisGnTAQFactor() returns the current "disregard TAQ" factor.

```
double          getDisSnTAQFactor ()
SNReturnCode    setDisSnTAQFactor(double aFactor)
```

Gets or sets the surname "disregard TAQ" factor. The "disregard TAQ" factor is applied to a segment score when both segments have one or more associated disregard TAQs, but no disregard TAQ value is common to both segments. This factor should be viewed as a penalty that gets applied to the segment score in the situation described above. See the discussion on TAQs for an explanation of the different types of TAQ values. See the discussion on TAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aFactor   A double value between 0.0 and 1.0 inclusive.

Return Values:

setDisSnTAQFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_DIS_SN_TAQ_FACTOR	The specified factor is invalid.

getDisSnTAQFactor() returns the current "disregard TAQ" factor.

```
double          getGnAnchorFactor ()
SNReturnCode    setGnAnchorFactor(double aFactor)
```

Gets or sets the factor to apply to agiven name segment score when the two segments are in place, but their ordinal position is not the anchor segment (as specified with the setGnAnchorSegmentMode() method).

The anchor factor should be viewed as a way to diminish the importance of a match if the match occurs between two segments that are not in the anchor segment position. For example, Arabic given names commonly include one or more segments. The first segment is the more stable segment and should therefore be considered the anchor segment. A match between two segments in the second given name position is considered to be of less importance (relative to the first segment), and as such, that segment score is diminished by applying the anchor factor.

Note that the given name anchor factor is only applied when the two segments are in place (they are in the same position). Given name segments that are out of place are adjusted by the given name "out of place segment" score getGnOOPSFactor()). In addition, the given name anchor factor is only applied when the given name anchor segment mode (setGnAnchorSegmentMode()) has been set.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnAnchorFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_GN_ANCHOR_FACTOR	The specified factor is invalid.

getGnAnchorFactor() returns the current "given name anchor segment" factor.

```
SNAnchorSegMode    getGnAnchorSegmentMode ()
void               setGnAnchorSegmentMode(SNAnchorSegMode anAnchorMode)
```

Gets or sets the given name anchor segment mode. Setting the anchor segment mode causes the API to place emphasis on a particular segment within the given name (the first segment, or the last segment). When this feature is turned off, all segments are considered to be equally important. See thesetGnAnchorFactor() method for details on how the anchor segment affects segment scoring.

The given name anchor segment is also used to determine how segments in two names are lined up (to determine which segments are in place or out of place). When the anchor segment is set to SN_ANCHOR_SEG_NONE or SN_ANCHOR_SEG_FIRST, segment alignment starts from the left (the first segment). When the anchor segment is set to SN_ANCHOR_SEG_LAST, segment alignment starts from the right (the last segment). See thesetGnOOPSFactor() method for details on how the API adjusts the score of segments that are out of place.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

anAnchorMode	A SNAnchorSegMode value:	
	SN_ANCHOR_SEG_NONE	No segment carries more importanc another. Name segments are lined u left to determine which segment co are in place.
	SN_ANCHOR_SEG_FIRST	The first segment is the most impor segment. Name segments are lined left to determine which segment co are in place.
	SN_ANCHOR_SEG_LAST	The last (right most) is the most im segment. Name segments are lined right to determine which segment comparisons are in place.

Return Values:

getGnAnchorSegmentMode() returns the current "given name anchor segment" mode.

```
double          getGnCompressedNameScore ()
SNReturnCode    setGnCompressedNameScore(double aScore)
```

Gets or sets the score to assign to a successfulgiven name compressed name comparison. See the setCheckGnCompressedName() method for detail on compressed name comparisons.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aScore` A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnCompressedNameScore() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was succ
SN_INVALID_GN_COMPRESSED_NAME_SCORE	The specified score is inval

getGnCompressedNameScore() returns the current "given name compressed name" score.

```
double          getGnOOPSFactor ()
SNReturnCode    setGnOOPSFactor(double aFactor)
```

Gets or sets the given name "out of place segment" factor. This is the factor that is applied to a segment score when the two segments are out of place (their ordinal positions are different).The given name anchor segment mode (setGnAnchorSegMode()) affects how segment alignment is performed.

To understand how alignment affects in place/out of place determination, consider the given names "Earl Bob" and "James Earl Bob". If we align these names on the left, we get:

Name 1:	Earl	Bob	
Name 2:	James	Earl	Bob

If we line the names up on the right, we get:

Name 1:		Earl	Bob
Name 2:	James	Earl	Bob

Notice that in the first case, the "Earl" and "Bob" segments are out of place, so we would apply the given name "out of place segment" factor to their segment scores. In the second case, because we align on the right, the segments are in place, so their segment scores are not adjusted.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setGnOOPSFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_GN_OOPS_FACTOR	The specified factor is invalid.

getGnOOPSFactor() returns the current given name "out of place segment" factor.

```
SNTAQProcessingMode    getGnTAQProcessingMode ()
void                   setGnTAQProcessingMode(SNTAQProcessingMode aMode)
```

Gets or sets the mode that determines how to processgiven name TAQ values.

The following modes are supported:.

Mode	Description
SN_TAQ_MODE_IGNORE	The API will not check given name segments t are TAQ values.
SN_TAQ_MODE_JUST_REMOVE	The API will check each given name segment t a TAQ value. If so, the value is removed as tho existed.
SN_TAQ_MODE_IGNORE	The API will check each given name segment t a TAQ value. If so, the segment gets associated proper stem segment, and is used in the compu stem segment's score.

See the discussion onTAQs for an explanation of the different types of TAQ values. See the discussion onTAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aMode An SNTAQProcessingMode value of either SN_TAQ_MODE_IGNORE or SN_TAQ_MODE_JUST_REMOVE.

Return Values:

getGnTAQProcessingMode() returns the current given name TAQ processing mode.

```
double          getSnAnchorFactor ()
SNReturnCode    setSnAnchorFactor(double aFactor)
```

Gets or sets the factor to apply to asurname segment score when the two segments are in-place, but their ordinal position is not the anchor segment (as specified with the setSnAnchorSegmentMode() method).

The anchor factor should be viewed as a way to diminish the importance of a match if the match occurs between two segments that are not in the anchor segment position. For example, Hispanic surnames commonly include two segments. The first segment is the true surname and should therefore be considered the anchor segment. A match between two segments in the second position is considered to be of less importance (relative to the first segment), and as such, that segment score is diminished by applying the anchor factor.

Note that the surname anchor factor is only applied when the two segments are in place (they are in the same position). Surname segments that are out of place are adjusted by the surname "out of place segment" score (setSnOOPSFactor()). In addition, the surname anchor factor is only applied when the surname anchor segment mode (setSnAnchorSegmentMode()) has been set.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnAnchorFactor() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_SN_ANCHOR_FACTOR	The specified factor is invalid.

getSnAnchorFactor() returns the current "surname anchor segment" factor.

```
SNAnchorSegMode    getSnAnchorSegmentMode ()
void               setSnAnchorSegmentMode(SNAnchorSegMode anAnchorMode)
```

Gets or sets the surname anchor segment mode. Setting the anchor segment mode causes the API to place emphasis on a particular segment within the surname (the first segment, or the last segment). When this feature is turned off, all segments are considered to be equally important. See thesetSnAnchorFactor() method for details on how the anchor segment affects segment scoring.

The surname anchor segment is also used to determine how segments in two names are lined up (to determine which segments are in place or out of place). When the anchor segment is set to SN_ANCHOR_SEG_NONE or SN_ANCHOR_SEG_FIRST, segment alignment starts from the left (the first segment). When the anchor segment is set to SN_ANCHOR_SEG_LAST, segment alignment starts from the right (the last segment). See thgetSnOOPSFactor() method for details on how the API adjusts the score of segments that are out of place.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

anAnchorMode	A SNAnchorSegMode value:	
	SN_ANCHOR_SEG_NONE	No segment carries more importance than another. Name segments are line up on the left to determine which segment comparisons are in place.
	SN_ANCHOR_SEG_FIRST	The first segment is the most importan segment. Name segments are lined up the left to determine which segment comparisons are in place.

SN_ANCHOR_SEG_LAST	The last (right most) is the most important segment. Name segments ar lined up on the right to determine whi segment comparisons are in place.

Return Values:

getSnAnchorSegmentMode() returns the current "surname anchor segment" mode.

```
double          getSnCompressedNameScore ()
SNReturnCode    setSnCompressedNameScore(double aScore)
```

Gets or sets the score to assign to a successfulsurname compressed name comparison. See the setCheckSnCompressedName() method for detail on compressed name comparisons.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aScore	A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnCompressedNameScore() returns anSNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was succe

SN_INVALID_SN_COMPRESSED_NAME_SCORE The specified score is inval

getSnCompressedNameScore() returns the current "surname compressed name" score.

```
double          getSnOOPSFactor()
SNReturnCode    setSnOOPSFactor(double aFactor)
```

Gets or sets the surname "out of place segment" factor. This is the factor that is applied to a segment score when the two segments are out of place (their ordinal positions are different).The surname anchor segment mode getSnAnchorSegMode()) affects how segment alignment is performed.

To understand how alignment affects in place/out of place determination, consider the surnames "Garcia Gomez " and "Valdez Garcia Gomez". If we align these names on the left, we get:

Name 1:	Garcia	Gomez	
Name 2:	Valdez	Garcia	Gomez

If we line the names up on the right, we get:

Name 1:		Garcia	Gomez
Name 2:	Valdez	Garcia	Gomez

Notice that in the first case, the "Garcia" and "Gomez" segments are out of place, so we would apply the surname "out of place segment" factor to their segment scores. In the second case, because we align on the right, the segments are in place, so their segment scores are not adjusted.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aFactor`	A double value between 0.0 and 1.0 inclusive.

Return Values:

setSnOOPSFactor() returns an SNReturnCode value indicating the success of the operation:

SN_SUCCESS	The modification was successful.
SN_INVALID_SN_OOPS_FACTOR	The specified factor is invalid.

getSnOOPSFactor() returns the current surname "out of place segment" factor.

```
SNTAQProcessingMode    getSnTAQProcessingMode ()
void                   setSnTAQProcessingMode(SNTAQProcessingMode aMode)
```

Gets or sets the mode that determines how to processsurname TAQ values.

The following modes are supported:

Mode	Description
SN_TAQ_MODE_IGNORE	The API will not check surname segments to s TAQ values.
SN_TAQ_MODE_JUST_REMOVE	The API will check each surname segment to s TAQ value. If so, the value is removed as thou existed.
SN_TAQ_MODE_IGNORE	The API will check each surname segment to s TAQ value. If so, the segment gets associated ,proper stem segment, and is used in the compu stem segment's score.

See the discussion onTAQs for an explanation of the different types of TAQ values. See the discussion onTAQ Scoring for information on how TAQs are used to adjust segment scores.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aMode	An SNTAQProcessingMode value of either SN_TAQ_MODE_IGNORE or SN_TAQ_MODE_JUST_REMOVE.

Return Values:

getSnTAQProcessingMode() returns the current surname TAQ processing mode.

```
BOOL      getUseGnLeftBias()
void      setUseGnLeftBias(BOOL aBool)
```

Gets or sets the flag that determines ifgiven name segment comparisons should be biased towards matches that occur at the beginning of the segment. When this feature is turned on, as we move to the right, matching character pairs are given decreasingly less credit in calculating a segment score. When this feature is turned off, all matching character pairs receive full credit, regardless of their position with their respective segment.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aBool	A BOOL value of TRUE or FALSE.

Return Values:

getUseGnLeftBias() returns the current value of the flag (TRUE or FALSE).

```
BOOL    getUseGnVariants()
void    setUseGnVariants(BOOL aBool)
```

Gets or sets the flag that determines if given name segment comparisons should check to see if the two segments are linguistic variants of each other.

The API maintains internal tables that describe relationships between name variants. Each variant relationship has an associated score and culture. When comparing two segments, the API examines the value of the "use given name variants" flag. If it is turned on, the internal variant tables are searched to see if there is a variant relationship between the two segments, within the culture associated with this query (as determined by the SNQueryParms object used to perform the comparison). There is also a generic set of variants that are searched independent of culture. If a variant relationship is found, its associated score is assigned to the segment score, and no character based comparison is performed.

At present, the set of variants and their associated scores can not be modified by the developer.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getUseGnVariants() returns the current value of the flag (TRUE or FALSE).

```
BOOL      getUseSnLeftBias()
void      setUseSnLeftBias(BOOL aBool)
```

Gets or sets the flag that determines ifsurname segment comparisons should be biased towards matches that occur at the beginning of the segment. When this feature is turned on, as we move to the right, matching character pairs are given decreasingly less credit in calculating a segment score. When this feature is turned off, all matching character pairs receive full credit, regardless of their position with their respective segment.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

`aBool` A BOOL value of TRUE or FALSE.

Return Values:

getUseSnLeftBias() returns the current value of the flag (TRUE or FALSE).

```
BOOL    getUseSnVariants()
void    setUseSnVariants(BOOL aBool)
```

Gets or sets the flag that determines ifsurname segment comparisons should check to see if the two segments are linguisticvariants of each other.

The API maintains internal tables that describe relationships between name variants. Each variant relationship has an associated score and culture. When comparing two surname segments, the API examines the value of the "use surname variants" flag. If it is turned on, the internal variant tables are searched to see if there is a variant relationship between the two segments, within the culture associated with this query (as determined by the SNQueryParms object used to perform the comparison). There is also a generic set of variants that are searched independent of culture. If a variant relationship is found, its associated score is assigned to the segment score, and no character based comparison is performed.

At present, the set of variants and their associated scores can not be modified by the developer.

These are advanced methods and should only be used by those with a deep understanding of name searching issues.

Parameters:

aBool A BOOL value of TRUE or FALSE.

Return Values:

getUseSnVariants() returns the current value of the flag (TRUE or FALSE).

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNResultsList Class

- Class Overview
- Methods Summary
- Attributes
- Construction
- Method Details

Overview

The SNResultList class provides a mechanism to manage the results of a query. Specifically, an SNResultsList object handles issues of comparing and sorting evaluation names (SNEvalNameData objects) that have been determined to be matches. In addition, an SNResultsList object can trim the set of matching names down to the best N names, where N is specified by the developer.

An SNResultsList object is equipped to manage a single query session (a set of comparisons between a single query name and one or more evaluation names). To use an SNResultsList object, the developer must create a new object via the SNResultsList() constructor, and attach it (via the SNQueryNameData::setResultsList() method) to the query name (SNQueryNameData) whose results it should manage. As calls are made to SNEvalNameData's performComp() method, the SNResultsList will manage those evaluation names that are considered matches (as determined by the SNEvalNameData::getCompResult() method). After all evaluation names have been compared to the query, the developer can interrogate the SNResultsList object to determine the number of matches, and request pointers to matches themselves (pointers to SNEvalNameData objects). After all matches have been processed. the developer should delete the SNResultsList object. A new query should create a new SNResultsList object, rather than reuse an existing one.

SNResultsList provides two important management functions. First, it sorts matching SNEvalNameData objects automatically. Sorting is accomplished by invoking the SNEvalNameData::compareScore() method to determine which matches are better than others. This provides the developer with a great deal of flexibility, because the compareScore() method can be overridden to allow for customized sorting behavior. The default method provides a robust set of comparison criteria, but the developer can alter the functionality, or incorporate new application specific data into the comparison.

The second important management function is that of results trimming. When constructing an SNResultsList object, the developer can specify the maximum number of matches the results list should hold. As matches are added to the results list, only the requested number of matches are retained. The object ensures that the best matches (as determined by SNEvalNameData::compareScore()) remain in the results list, and it also handles the memory management associated with discarding those matches that are "squeezed out" by better matches.

Most applications will benefit from the functionality of SNResultsList. However, use of an SNResultsList object is optional. If desired, the developer can provide for their own match management. For example, an application may choose to examine the return code from SNEvalNameData's performComp() method directly, rather than depending on an SNResultsList object for sorting and filtering.

Methods Summary

Common Methods:

SNResultsList()	Constructor for the class.
addHit()	Adds a name object to the results list. Called by the API, not the developer.
getHitAt()	Returns the name object at the specified index. Used to retrieve matches at the end of a query session.
getNumHits()	Returns the number of name objects in the results list.
getStatus()	Returns the status of the results list object. Used for error checking and reporting.

Attributes:

All attributes within the SNResultsList class are protected, and not available to the developer.

Method Details:

Constructors:

```
SNResultsList(int maxHits);
```

Constructs a new results list. If *maxHits* is specified as a number, the list will contain up to *maxHits* matches at any given time. Alternatively, *maxHits* can be specified as the special constant SN_RESULTS_LIST_SIZE_EXPANDABLE, in which case the results list can grow to any size (within available memory limitations).

On successful construction, the new results list is empty, and the status of the object is set to SN_SUCCESS. Use the getStatus() member function to validate successful construction.

Parameters:

`maxHits` The maximum number of matches the results list can hold at any one time. A special value of SN_RESULTS_LIST_SIZE_EXPANDABLE indicates that the list should grow as needed. This parameter must be a number greater than 1, or the special constant SN_RESULTS_LIST_SIZE_EXPANDABLE.

Return Values:

None. However, the getStatus() member should be called to validate successful construction.

Memory Management:

The responsibility of deleting an SNResultsList object lies with the developer. In general, an SNResultsList object should be deleted at the end of the query session. after all results have been retrieved and processed.

Because SNResultsList makes copies of the SNEvalNameData objects it manages, the developer may delete the SNEvalNameData objects immediately after calling SNEvalNameData's performComp() method.

Examples:

```
The example below shows the construction of an SNResultsList object,
and its use in a query session:


SNEvalNameData   *candidate1;
SNEvalNameData   *candidate2;
SNQueryNameData  *queryName;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode     retCode;
SNResultsList    *myResultsList = NULL;

candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
candidate2 = new SNEvalNameData(queryParms, "Earl", "Jhonas");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");


queryName->setResultsList(myResultsList);

candidate1->performComp(queryName);
candidate2->performComp(queryName);

// eval names can be deleted after being compared to the query
delete candidate1;
delete candidate2;

if (myResultsList->getNumHits() > 0)     {
   SNEvalNameData *matchName = myResultsList->getHitAt(0);
   printf("best match was %s, %s\n", matchName->getSn(), matchName->getGn(
}
else
   printf("Neither name Matched");

delete myResultsList;
delete queryName;
```

SNReturnCode addHit(SNEvalNameData *aHit)

Adds an evaluation name object (SNEvalNameData) to the results list. This method is invoked by the API, and not by the developer. Specifically, it is called during SNEvalNameData's performComp() method, when an evaluation name is determined to be a match. This method makes a copy of the name object, and assumes responsibility for its deletion.

Parameters:

`aHit`	A pointer to the SNEvalNameData object that should be added to the results list.

Return Values:

An SNReturnCode value indicating the success or failure of the operation:

SN_SUCCESS:	The operation was successful.
SN_RESULTS_LIST_INSERT_ALLOC_FAILURE:	A memory allocation problem occurred.
SN_RESULTS_ARRAY_NULL_ERROR:	A memory allocation problem occurred.

`SNEvalNameData * getHitAt(int anIndex)`

Returns a pointer to the SNEvalNameData object at the specified index. The index is 0 based, so getHitAt(0) returns a pointer to the best match.

If *anIndex* specifies an index that is out of range, the function returns NULL. Applications generally first call getNumHits() to determine the valid range of index values that can be supplied to this method.

The SNResultsList object owns the objects it maintains. As evaluation names are added to the results list, trimming might occur, which can result in the deletion of SNEvalNameData objects that get "squeezed out" to make room for better matches. As a result, an application should not rely on the validity of a pointer obtained by getHitAt() after a subsequent call to SNEvalNameData::performComp(). Similarly, because SNResultsList's destructor deletes the objects it manages, all pointers obtained by calls to getHitAt() become invalid once the results list is deleted.

Parameters:

`anIndex`	The 0 based index of the desired evaluation name object.

Return Values:

A pointer to the SNEvalNameData object at the specified index. If the specified index is out of range, the function returns NULL.

Examples:

```
The example below shows a sample query session using an SNResultsList
object. Notice that we call getHitAt() to retrieve the best match:


SNEvalNameData    *candidate1;
SNEvalNameData    *candidate2;
SNQueryNameData   *queryName;
SNQueryParms      *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode      retCode;
SNResultsList     *myResultsList = NULL;

candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
candidate2 = new SNEvalNameData(queryParms, "Earl", "Jhonas");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

myResultsList = new SNResultsList(1);    // create a manager for just 1 mat
queryName->setResultsList(myResultsList);

candidate1->performComp(queryName);
candidate2->performComp(queryName);

delete candidate1;
delete candidate2;

if (myResultsList->getNumHits() > 0)    {

   printf("best match was %s, %s\n", matchName->getSn(), matchName->getGn(
}
else
   printf("Neither name Matched");

delete myResultsList;
delete queryName;
```

`int getNumHits()`

Returns the number of matches in the results list. This is NOT the number of matches the list is capable of holding, but is the number of matches available for the user to retrieve via the getHitAt() method.

Parameters:

None.

Return Values:

The number of matches in the results list.

Examples:

```
The example below shows a sample query session using an SNResultsList
object. Notice that we call getNumHits() to make sure we have a hit
to retrieve:


SNEvalNameData   *candidate1;
SNEvalNameData   *candidate2;
SNQueryNameData  *queryName;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode     retCode;
SNResultsList    *myResultsList = NULL;

candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
candidate2 = new SNEvalNameData(queryParms, "Earl", "Jhonas");
queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

myResultsList = new SNResultsList(1);   // create a manager for just 1 mat
queryName->setResultsList(myResultsList);

candidate1->performComp(queryName);
candidate2->performComp(queryName);

delete candidate1;
delete candidate2;


   SNEvalNameData *matchName = myResultsList->getHitAt(0);
   printf("best match was %s, %s\n", matchName->getSn(), matchName->getGn(
}
else
   printf("Neither name Matched");

delete myResultsList;
delete queryName;
```

`SNReturnCode getStatus()`

Returns the status of the results list object. This method is for error checking purposes, and is usually called after an attempt to construct an SNResultsList object.

Parameters:

None.

Return Values:

An SNReturnCode value indicating the status of the object:

SN_SUCCESS:	Construction was successful.
SN_RESULTS_LIST_ALLOCATION_ERROR:	A memory allocation problem occurre
SN_INVALID_RESULTS_LIST_SIZE:	An invalid results list size was specifi results list size specifier must be a nu greater than 1, or the special constant SN_RESULTS_LIST_SIZE_EXPAN

Examples:

The example below shows a sample query session using an SNResultsList object. Notice that we call getStatus() to make sure our results list was created properly:

```
SNEvalNameData   *candidate1;
SNEvalNameData   *candidate2;
SNQueryNameData  *queryName;
SNQueryParms     *queryParms = new SNQueryParms(SN_PARMS_GENERIC);
SNReturnCode     retCode;
SNResultsList    *myResultsList = NULL;

myResultsList = new SNResultsList(1);   // create a manager for just 1 mat

   candidate1 = new SNEvalNameData(queryParms, "Bob Earl", "Jones");
   candidate2 = new SNEvalNameData(queryParms, "Earl", "Jhonas");
   queryName = new SNQueryNameData(queryParms, "James Earl", "Jones");

   queryName->setResultsList(myResultsList);

   candidate1->performComp(queryName);
   candidate2->performComp(queryName);

   delete candidate1;
   delete candidate2;

   if (myResultsList->getNumHits() > 0) {
      SNEvalNameData *matchName = myResultsList->getHitAt(0);
      printf("best match was %s, %s\n", matchName->getSn(), matchName->get
   }
   else
      printf("Neither name Matched");
   delete queryName;
}
else
   printf("Error creating results list\n");

delete myResultsList;
```

SNAPI is a trademark of Language Analysis Systems. All other products mentioned are registered trademarks or trademarks of their respective companies.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday January 23, 1998.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNAPI Functions Documentation

The SNAPI API provides a small number of functions that the developer may find useful.

SN_get_error_text()	Returns the message text for a specified SNAPI return code.
SN_shutdown()	Releases global resources allocated by the SNAPI system.
SN_startup()	Forces allocation of SNAPI global resources. If this function is not called, the resources will be allocated during the first query.
SN_strip()	A utility function to remove leading and trailing white space from a NULL terminated string.
SN_strrchr()	A utility function that searches backwards in a string for a specified character. Differs from strrchr() in that the string does not have to be NULL terminated.

```
void SN_get_error_text(SNReturnCode errorCode, char *textBuffer, int maxChars)
```

Retrieves the message text associated with a SNAPI return code. See the associated documentation on SNReturnCode for a list of possible error codes.

Parameters:

`errorCode`	The SNReturnCode value for which text is to be retrieved.
`textBuffer`	A buffer to hold the message text.
`maxChars`	The size of textBuffer (minus 1 for the NULL terminator).

Return Values:

None. On return, textBuffer contains the message text.

Examples:

```
The example below shows a failed attempt to create an SNResultsList,
and a subsequent call to SN_get_error_text():
```

```
SNResultsList   *myResultsList = NULL;
SNReturnCode    retCode;

myResultsList = new SNResultsList(-4);   // <--  invalid call
retCode = myResultsList->getStatus();

if (retCode != SN_SUCCESS)   {
   char msgBuf[1000 + 1];


   printf("Problem creating results list, msg was: %s\n", msgBuf);
}
else
  printf("Results list created OK.");

delete myResultsList;
```

void SN_shutdown()

Releases global resources that have been allocated by the SNAPI API.

The SNAPI API uses several lookup tables and similar resources. When the application exits, the operating system releases these resources, as it does with all resources associated with the process. However, many debugging environments check for memory leaks just before an application exits. To prevent debugging messages of this nature, call the SN_shutdown() function just before your application exits.

Parameters:

None.

Return Values:

None.

void SN_startup()

Allocates the global resources required by the SNAPI API.

The SNAPI API uses several lookup tables and similar resources. Each time one of these resources is needed, a check is made to see if the resource has been created. If not. the resource is created at that time. This may result in a slight delay the first time a resource is required.

By calling the SN_startup() function, all global resource allocation can be controlled by the developer.

Parameters:

None.

Return Values:

None.

□ `void SN strip(char *aString)`

Alters the supplied string by removing leading and trailing whitespace.

Parameters:

`aString` A NULL terminated string.

Return Values:

None. On return, aString has been stripped.

□ `char * SN_strrchr(char *stringStart, char *searchPos, char searchChar)`

Searches backwards through a string, looking for the specified search character. This function is similar to strrchr(), except that we also specify the position in the string to start searching, rather than assuming the string is NULL terminated and starting at the end.

If the search character is not found (we reach *stringStart* without finding the search character), we return NULL. Otherwise, we return a pointer to the first occurrence of search character we come across.

Parameters:

`stringStart`	A pointer to the start of the string to be searched.
`searchPos`	A pointer to the character to begin the reverse search. This should point to a character to the right of the stringStart.
`searchChar`	The character we are trying to find.

Return Values:

A pointer to the first occurrence of *searchChar* we find in our reverse search, or NULL if we reach *stringStart* without finding *searchChar*.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday January 23, 1998.

SNAPI Developer Support

SNAPI Data Types and Enumerations Documentation

The SNAPI API defines a number of data types and enumerations that are used throughout the system:

Data Types

BOOL A Boolean data type. The variable may.take the pre-defined values TRUE or FALSE. This data type is provided to appease C++ compilers that do not yet support the standard bool data type.

word A short int.

Enumerated Types

SNNameFormat Indicates the format of a single string name when constructing a name. See the SNEvalNameData::SNEvalNameData() and SNQueryNameData::SNQueryNameData() constructors for details on how these values are interpreted. Possible values are listed below.

SN_SURNAME_COMMA_GIVENNAME	The name is of the form "surname, given name"
SN_LAST_SEG_IS_SURNAME	The last (right most) segment is the surname. TAQ values are removed from consideration when determining the last segment.
SN_NAME_FORMAT_UNKNOWN	The name format is unknown.

SNParmsType

Specifies a particular parameter type when constructing a new SNQueryParms object. See the SNQueryParms::SNQueryParms() constructor for additional details. Possible values are listed below.

SN_PARMS_GENERIC	Specifies a set of parameters appropriate for searching Anglo (English) names, or names of unknown or mixed ethnicity.
SN_PARMS_ARABIC	Specifies a set of parameters appropriate for searching Arabic names.
SN_PARMS_CHINESE	Specifies a set of parameters appropriate for searching Chinese names.
SN_PARMS_HISPANIC	Specifies a set of parameters appropriate for searching Hispanic names.
SN_PARMS_KOREAN	Specifies a set of parameters appropriate for searching Korean names.
SN_PARMS_RUSSIAN	Specifies a set of parameters appropriate for searching Russian names.

SNSegScoreMode

Specifies the segment score mode when adjusting an SNQueryParms object via the setGnSegmentScoreMode() or setSnSegmentScoreMode() methods. See either of these methods for additional details. Possible values are listed below.

SN_SEGMODE_HIGHEST	The score assigned to the name field is the highest score found when comparing the segments.
SN_SEGMODE_LOWEST	The score assigned to the name field is the best low score found when comparing the segments.
SN_SEGMODE_AVG	The score assigned to the name field is the best average score found when comparing the segments.

SNTAQProcessingMode

Specifies how to handle TAQ processing when adjusting an SNQueryParms object via the setGnTAQProcessingMode() or setSnTAQProcessingMode(). See either of these methods for additional details. Possible values are listed below.

SN_TAQ_MODE_IGNORE	The API will not perform any TAQ processing on the name field.
SN_TAQ_MODE_JUST_REMOVE	The API will remove any TAQ values from the name field, but will not adjust any scores.
SN_TAQ_MODE_IGNORE	The API will remove any TAQ values from the name field, and will segment scores accordingly.

SNAnchorSegMode

Specifies the anchor segment when adjusting an SNQueryParms object via the setGnAnchorSegmentMode() or setSnAnchorSegmentMode() methods. See either of these methods for additional details. Possible values are listed below.

SN_ANCHOR_SEG_NONE	No segment carries more importance than another. Name segments are lined up on the left to determine which segment comparisons are in place.
SN_ANCHOR_SEG_FIRST	The first segment is the most important segment. Name segments are lined up on the left to determine which segment comparisons are in place.
SN_ANCHOR_SEG_LAST	The last (right most) segment is the most important segment. Name segments are lined up on the right to determine which segment comparisons are in place.

SNReturnCode

A set of return codes. Each return code specifies a particular condition within the API. Many functions within the API return a variable of type SNReturnCode. The global function SN_get_error_text() can be used to retrieve a textual description of the code. Each code's meaning is also documented below.

SN_SUCCESS	Operation was successful.
SN_MATCH	The comparison resulted in a match.

SN_NO_MATCH	The comparison did not result in a match.
SN_INVALID_SCORE_THRESH	Bad score threshold value.
SN_INVALID_GN_INIT_SCORE	Bad given name Initial score.
SN_INVALID_SN_INIT_SCORE	Bad surname Initial score.
SN_INVALID_GN_INIT_ON_INIT_MATCH_SCORE	Bad given name "Exact Initial match" score.
SN_INVALID_SN_INIT_ON_INIT_MATCH_SCORE	Bad surname "Exact Initial match" score.
SN_INVALID_NFN_SCORE	Bad "No First Name" score.
SN_INVALID_FNU_SCORE	Bad "First Name Unknown" score.
SN_INVALID_NLN_SCORE	Bad "No Last Name" score.
SN_INVALID_LNU_SCORE	Bad "Last Name Unknown" score.
SN_INVALID_GN_ANCHOR_FACTOR	Bad given name anchor factor.
SN_INVALID_SN_ANCHOR_FACTOR	Bad surname anchor factor.
SN_INVALID_GN_OOPS_FACTOR	Bad given name "Out of Place Segment" factor.

SN_INVALID_SN_OOPS_FACTOR	Bad surname "Out of Place Segment" factor.
SN_INVALID_ABS_DEL_GN_TAQ_FACTOR	Bad given name "absent delete TAQ" factor.
SN_INVALID_ABS_DEL_SN_TAQ_FACTOR	Bad surname "absent delete TAQ" factor.
SN_INVALID_ABS_DIS_GN_TAQ_FACTOR	Bad given name "absent disregard TAQ" factor.
SN_INVALID_ABS_DIS_SN_TAQ_FACTOR	Bad surname "absent disregard TAQ" factor.
SN_INVALID_DEL_GN_TAQ_FACTOR	Bad given name "delete TAQ" factor.
SN_INVALID_DEL_SN_TAQ_FACTOR	Bad surname "delete TAQ" factor.
SN_INVALID_DIS_GN_TAQ_FACTOR	Bad given name "disregard TAQ" factor.
SN_INVALID_DIS_SN_TAQ_FACTOR	Bad surname "disregard TAQ" factor.
SN_INVALID_GN_COMPRESSED_NAME_SCORE	Bad given name "compressed name" score.
SN_INVALID_SN_COMPRESSED_NAME_SCORE	Bad surname "compressed name" score.
SN_RESULTS_LIST_INSERT_ALLOC_FAILURE	Could not alloc space for new hit in the results list.

SN_GN_VAR_TABLE_CREATION_ERROR	Could not create GN variant table.
SN_SN_VAR_TABLE_CREATION_ERROR	Could not create SN variant table.
SN_TAQ_TABLE_CREATION_ERROR	Could not create TAQ table.
SN_SEG_BREAK_CHARS_CREATION_ERROR	Could not create seg break chars string.
SN_NOISE_CHARS_CREATION_ERROR	Could not create noise chars string.
SN_INVALID_RESULTS_LIST_SIZE	Invalid size requested for results list.
SN_RESULTS_LIST_ALLOCATION_ERROR	Could not allocate initial space for results list.
SN_RESULTS_ARRAY_NULL_ERROR	The internal results list array is NULL.
SN_TAQ_RECORD_ALLOC_ERROR	Problem allocating space for a new TAQ record.
SN_VARIANT_ALLOC_ERROR	Problem allocating space for a new variant record.
SN_VARIANTS_DONT_EXIST	An attempt was made to alter the score of a relationship that did not already exist.

SN_INVALID_VARIANT_SCORE	An invalid score was specified for a variant relationship.
SN_TOO_MANY_VARIANTS_FOR_NAME	The maximum number of variants per name has been exceeded for a name.
SN_VARIANT_ALREADY_RELATED	An attempt was made to relate two names that were already related.
SN_PARMS_FILE_OPEN_ERROR	Problem opening a parms file
SN_PARMS_FILE_NOISE_CHARS_ERROR	Problem reading noise chars from a parameters file.
SN_PARMS_FILE_BREAKS_CHARS_ERROR	Problem reading break chars from parameters file.
SN_TAQ_NOT_FOUND	The specified TAQ could not be found.
SN_TAQ_ALREADY_EXISTS	The specified TAQ is already defined.
SN_INVALID_GN_THRESH	The specified GN Thresh is invalid.

SN_INVALID_SN_THRESH	The specified SN Thresh is invalid.
SN_INVALID_GN_WEIGHT	The specified GN Weight is invalid.
SN_INVALID_SN_WEIGHT	The specified SN Weight is invalid.
SN_INVALID_CULTURE_CODE	The specified Culture Code is invalid.
SN_ERROR_READING_CUSTOM_ PARAMETER_FROM_FILE	An error occurred while reading a custom parameter from a file.
SN_ERROR_WRITING_CUSTOM_ PARAMETER_TO_FILE	An error occurred while writing a custom parameter to a file.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Monday December 01, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNAPI Constants Documentation

The SNAPI API defines a number of constants that are used throughout the system:

<table>
<tr><th>Name</th><th>Value</th><th>Description</th></tr>
<tr><td>EOS</td><td>'\0'</td><td>The end of string marker.</td></tr>
<tr><td>FALSE</td><td>0</td><td>Boolean constant.</td></tr>
<tr><td>SN_DEFAULT_NOISE_CHARS</td><td>"!\"#$%()*+./:;<=>?@[\\}'{|}`123456789"</td><td>The default set of characters that are treated as though they did not exist when found in a name.</td></tr>
<tr><td>SN_DEFAULT_SEG_DELIM_CHARS</td><td>"- \t"</td><td>The default set of characters that act as segment delimiters.</td></tr>
<tr><td>SN_DEFAULT_WHITESPACE</td><td>" \n\r\t"</td><td>The set of characters considered to be whitespace.</td></tr>
<tr><td>SN_MAX_GN_LEN</td><td>255</td><td>Max length of the given name. If a given name is longer, truncation occurs.</td></tr>
<tr><td>SN_MAX_MN_LEN</td><td>255</td><td>Max length of the middle name. If a middle name is longer, truncation occurs.</td></tr>
<tr><td>SN_MAX_SEG_LENGTH</td><td>30</td><td>Max length of a single name segment. If a segment is longer, truncation occurs.</td></tr>
<tr><td>SN_MAX_SEGS_AFTER_TAQ</td><td>5</td><td>Max number of segments per name field after TAQ removal. Any segments after the maximum are disregarded.</td></tr>
</table>

SN_MAX_SEGS_BEFORE_TAQ	10	Max number of segments per name field before TAQ removal. Any segments after the maximum are disregarded.
SN_MAX_SN_LEN	255	Max length of the surname. If a surname is longer, truncation occurs.
SN_MAX_TAQS_PER_SEGMENT	5	Max number of TAQs that can be associated with one segment. Any TAQs over the maximum are disregarded.
SN_RESULTS_LIST_SIZE_EXPANDABLE	-1	Specifies that the results list should grow as needed.
TRUE	1	Boolean constant.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Tuesday November 25, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

SNAPI Vocabulary and Terms

In talking about names and name searching, a definition of some vocabulary can be helpful. Below, we present a brief discussion of some of the terms and concepts found throughout the documentation:

Name	**Description**
Anchor Segment	The name segment within a name field that is considered the most important. Most cultures do not have an anchor segment, because all segments are considered equally important. However, some cultures place emphasis on a particular segment. For example, the first segment of an Arabic given name (e.g., Mohammaed bin Salam) is considered most important and the second segment of a Lusophone or Portuguese surname (e.g., Ferreira Dos Santos) is considered most important.
Given Name	The portion of a name that does NOT reference the family name. In Anglo (English) names, this is typically the first name.
Name Field	The given name or surname portion of a name, considered separately. The SNAPI name model currently uses just these two name fields.
Name Segment	Any portion of a name that is separated by a segment delimiter. The most common segment delimiter is a space. For example, the name "James Earl Jones" has three segments: "James", "Earl", and "Jones".
Name Variant	Linguistic variants include a wide variety of motivated variations of a name, including nicknames, abbreviations, phonetic variants, and cultural variants, among others. Common Anglo (English) nicknames include Jack/John and Bill/William. The SNAPI system currently uses a table of name variants organized by culture to provide special handling for these names.
Stem	A stem is a non-TAQ segment. Stem segments are considered to be part of the actual name, while TAQ values are adjuncts to the name. Stem segments receive segment scores, while TAQ segments do not (rather, they are used to adjust their associated stem's segment score).
Surname	The portion of a name that describe a person's family (i.e., family name). In Anglo (English) names, this is typically the last name.
TAQ	An acronym that stands for Titles, Affixes (prefixes and suffixes) and Qualifiers. TAQ values can be thought of as name modifiers. Common Anglo (English) examples include "Jr" and "Dr". The SNAPI system uses a table of TAQ values organized by culture to provide special handling for these modifiers. TAQ values are broken up into two groups. *Delete* TAQ values are modifiers that do not provide any true meaning regarding a person's identity. Examples of *Delete* TAQs include "Mr" and "Dr". *Disregard* TAQ values do provide extra information. Example of *Disregard* TAQs include "Jr" and "De". Each TAQ value is also classified as either a prefix or a suffix. This classification is used to determine the TAQ's associated stem segment.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday January 23, 1998.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

TAQ Scoring

TAQ processing is one of the most complicated aspects of the API. This section provides a condensed overview of how TAQ values are processed and used to adjust segment scores. In general, most applications do not need to be concerned with these advanced issues.

TAQ factors help address issues that arise when comparing names such as "James Brown Jr" and "James Brown Sr". The API has the ability to recognize TAQ values such as "Jr", "Sr", and "Dr". The API maintains an internal table of TAQ values, organized by culture (there is also a set of generic TAQ values that span all cultures). As a name is processed, each name segment is examined to determine if it is a TAQ value. The culture associated with the SNQueryParms object used to create the name object determines which culture's TAQ values should be considered (in addition to the generic set). Note that TAQ values are restricted to single segments.

Each TAQ value is classified as either a prefix or suffix in order to determine the stem segment with which a particular TAQ value should be associated. Once associated with a stem segment, the TAQ value is removed from the name (no segment score is generated for the TAQ value). However, after each remaining stem segment has received a score, its associated TAQ(s) are examined and scores are adjusted according to specific rules.

TAQ scoring occurs after each segment has received a segment score. The process of adjusting segment scores based on associated TAQ values is quite complicated for two reasons. First, each segment can have multiple associated TAQ values. Second, the associated TAQ values can be of mixed type (Disregard and/or Delete). The following table attempts to describe the algorithm employed when scoring associated TAQ values for two segments:

Step	Description
1	If neither segment has an associated disregard value, proceed to step 5.
2	If the same disregard value is associated with both segments, proceed to step 5.
3	If both names have at least one disregard value, but none are common to both, apply the "disregard TAQ" factor and stop.
4	If one name has one or more disregard values, but the other has none, apply the "absent disregard TAQ" factor and stop.
5	If neither segment has an associated delete value, stop (do not modify the segment score).
6	If the same delete value is associated with both segments, stop (do not modify the segment score).
3	If both names have at least one delete value, but none are common to both, apply the "delete TAQ" factor and stop.
4	If one name has one or more delete values, but the other has none, apply the "absent delete TAQ" factor and stop.

SNAPI is a trademark of Language Analysis Systems. All other products mentioned are registered trademarks or trademarks of their respective companies.

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Tuesday November 25, 1997.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

Programmer's Tutorial

Welcome to the programmer's tutorial. This is a good place to start if you have never worked with the API. We will present some very simple code examples here, and explain how they work. From here, you can check out the API Documentation or the FAQ for greater detail and discussions of more advanced topics.

Getting Started

To write a program using the API, you will need the following:

- A C++ compiler.
- The SNAPI header files (distributed with SNAPI).
- A library file appropriate for your compilation environment (distributed with SNAPI).

The header files are necessary so that your compiler is aware of the objects and functions that SNAPI provides. The library is necessary so that your linker can resolve the external references to the SNAPI objects and functions. In addition, you must instruct your compiler and linker as to the location of these files. This process is specific to each development environment. Please consult your compiler documentation for instructions.

A Simple Application

The following is a simple application that takes two names (specified on the command line), and performs a comparison between the two. We will assume the names are in "surname, given name" format. Following the source code, each line that uses something from SNAPI is explained in detail. Note that the line numbers at the beginning of each line are for illustration only.

```
1       #include        <snapi.h>

2       #include        <stdio.h>
3       #include        <stdlib.h>

4       int     main(int argc, char *argv[])
5       {
6          char                    *name1 = argv[1];
7          char                    *name2 = argv[2];
8          SNQueryParms            queryParms(SN_PARMS_GENERIC);
9          SNQueryNameData         queryNameObject = new SNQueryNameData(&queryParms, n
10         SNEvalNameData          evalNameObject = new SNEvalNameData(&queryParms, nam
11         SNReturnCode            retCode;

12         retCode = evalNameObject->performComp(queryNameObject);

13         if ((retCode == SN_MATCH) || (retCode == SN_NO_MATCH)        {
14            if (retCode == SN_MATCH)
15               printf("Names Matched\n");
16          else
17             printf("Names Did Not Match\n");
18          printf("Name Score was %f, GN Score was %f, SN score was %f\n",
                    evalNameObject->getNameScore(), evalNameObject->getGnScore(),
                    evalNameObject->getSnScore());
19       }
20     else  {
21          char errorBuffer[1000 + 1];

22          SN_get_error_text(retCode, errorBuffer, 1000);
23          printf("An error occurred\n");
24          printf("Error text is %s\n", errorBuffer);
25       }
26       delete queryNameObject;
27       delete evalNameObject;
28       exit(0);
29    }
30    // end of program
```

Line 1 includes the *snapi.h* header file. This file should be included in any source file that references SNAPI objects, functions or data types.

Lines 6 and 7 assign pointers to the names specified on the command line. This is done for clarity. Remember that this program obtains the names it will compare via command line arguments.

Line 8 creates a new SNQueryParms object. This object encapsulates all the parameters that control how names are processed, and how comparisons between names are performed. An **SNQueryParms** object is created by specifying an SNParmsType value, which identifies a particular culture. The resulting object contains parameters appropriate for the specified culture. Our application requests parameters for a generic search. A typical application might adjust a small number of these parameters. For simplicity, this application uses the default parameters. The application is responsible for deleting any **SNQueryParms** objects it creates. Because we have created our **SNQueryParms** object on the stack, it will automatically be deleted when the main() function exits.

Line 9 creates a new SNQueryNameData object. When creating the object, we must specify an **SNQueryParms**, a name string, and an SNNameFormat variable that tells the constructor how to interpret the name. Again, for our example, we are assuming a format of "surname, given name". **SNQueryNameData** also includes other constructors (e.g. one that retreives the given name and surname as separate variables). The **SNQueryParms** object tells the API how to process certain aspects of the name (e.g. name variants and TAQ values).

Line 10 creates a new SNEvalNameData object. This object is very similar to an **SNQueryNameData** object, but with a few important differences. First, an **SNEvalNameData** object includes a method to compare itself to an **SNQueryNameData** object. In addition, it defines score attributes to hold the results of such a comparison. Most true search applications will create one **SNQueryNameData** object (perhaps for a name keyed in by the user), and many **SNEvalNameData** objects (one for each name in a database to be searched). Our simple application only compares two names, so the distinction between the query name and evaluation name name is not as dramatic here. The **SNQueryParms** object that is used to create this object should be the same **SNQueryParms** object that was used to create the **SNQueryNameData** object above. In general, when comparing two name objects, both objects should be created using the same **SNQueryParms** object.

Line 11 defines an SNReturnCode variable. Many SNAPI functions return a value of this type.

Line 12 performs the actual comparison between the query name and the evaluation name. Note that we pass the query name as a parameter to the evaluation name's performComp() method (only **SNEvalNameData** defines a comparison function, **SNQueryNameData** does not). The **performComp()** method conducts a comparison according to the parameters specified in the **SNQueryParms** object use to create the evaluation name. It returns a value indicating if the names matched (**SN_MATCH**), did not match (**SN_NO_MATCH**) or if there was some sort of error (various other return codes). After the **performComp()** method returns, the **SNEvalNameData** object's score attributes are set. These attributes include the given name score, surname score, and overall name score.

Line 13 ensures that there were no problems in performing the comparison.

Lines 14 through 17 examine the **SNReturnCode** value and print out a message indicating whether the name is considered a match or not.

Line 18 prints out the scores that were computed during the **performComp()** method. It calls the functions getNameScore(), getGnScore() and getSnScore(). Each of these functions returns a value type double between 0.0 and 1.0 inclusive, where 1.0 represents an exact match.

Lines 20 through 25 handle the case where the comparison function produced some kind of error (an **SNReturnCode** value other than **SN_MATCH** or **SN_NO_MATCH**). Line 22 calls the function SN_get_error_text(), which retrieves the error text associated with a particular **SNReturnCode** value. We pass in a buffer to hold the error text, along with the size of our buffer.

Lines 26 and 27 delete the **SNQueryNameData** and **SNEvalNameData** objects that we created at the beginning of the program. Any SNAPI objects that are created by the developer must be deleted.

Another Example

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday January 23, 1998.

SNAPI Developer Support

[Welcome | Overview | What's New | FAQs | API Documentation | Tutorial | Sample Code | Bugs | Suggestions | Download | Search]

FAQ -- Frequently Asked Questions

This page contains answers to common questions. The FAQ is broken into two sections: A general FAQ, Developer's FAQ.

General Questions:

1. What is SNAPI?
2. What platforms does SNAPI support?
3. How much does it cost?
4. What kind of support/training options are available?
5. What kind of documentation is provided?
6. What is the current production release number for SNAPI?
7. Can SNAPI interface with my database (e.g. Oracle. Sybase. etc.).
8. What type of indexes does SNAPI support?
9. What performance benchmarks are available?
10. What are the machine resource requirements (Memory. disk space) for SNAPI?

Developer Questions:

1. Which header files should I include in my programs?
2. Does the API support mutely-threading?
3. How can I only look at surnames when doing a name comparison?
4. I have a middle name field in my database. How do I include this information in my name comparisons?
5. My application is written in (COBOL. FORTRAN. Java). How can I use SNAPI to add name checking to my app?
6. How do I associate other data (such as database record ids) with the name objects SNAPI uses? I need this so that when SNAPI tells me a name matches my query. I can look up other information associated with that name.
7. I have other information (such as age and social security number) that I want to include in the comparison process. How do I do this?
8. Which C++ compilers does SNAPI support?
9. My database contains names of mixed culture. Which culture should I specify when I create my parameters object?
10. My database has over 10 million names. Can SNAPI handle this kind of volume?
11. I am planning to create name objects for every name in my database, and hold them in memory, reusing them for each query. How much memory will I need to do this?
12. How can I change the scores assigned to a particular name variant association. How can I delete variant associations or add new ones?
13. How can I delete TAQ values or add new ones?
14. What types of name formats does SNAPI support?

1. What is SNAPI?

SNAPI is an application program interface that lets a developer add sophisticated name searching capabilities to an application. SNAPI is not an end product, but is a tool to integrate name searching and comparison capabilities into applications. SNAPI is a C++ API, and requires a C++ compiler.

Back to Top

---

2. What platforms does SNAPI support?

SNAPI works on any platform that supports a C++ compiler. LAS has tested the API on both Windows (95 and NT) and UNIX systems.

Back to Top

---

3. How much does it cost?

Pricing for SNAPI is dependent on a number of factors. Please contact LAS at info@las-inc.com for details.

Back to Top

---

4. What kind of support/training options are available?

A variety of training, support, and consulting options are available to licensees of SNAPI. Please contact LAS at info@las-inc.com to discuss your particular needs.

Back to Top

---

5. What kind of documentation is provided?

Documentation for SNAPI is provided in HTML format. The documentation includes an overview, a tutorial, and full documentation of the classes and functions that make up the API..

Back to Top

---

6. What is the current production release number for SNAPI?

SNAPI version 1.0 is currently in beta testing, with an expected release in Q1 1998.

Back to Top

---

7. Can SNAPI interface with my database (e.g. Oracle, Sybase, etc.).

SNAPI makes no assumptions about data storage mechanisms. Data access is the responsibility of the developer.

Back to Top

8. What type of indexes does SNAPI support?

Version 1.0 of SNAPI does not include any type of indexing support. When searching a database for a query name, all names are evaluated. However, a developer can often segment a database based on application specific rules. For example, a query that considers both name and age might only retrieve names from the database for people that are within 10 years of the age specified with the query.

The next version of SNAPI will include mechanisms to support one or more indexing strategies.

Back to Top

9. What performance benchmarks are available?

On an Intel Pentium 133 MHz machine, SNAPI performs roughly 10,000 name comparisons per second. Because SNAPI is CPU intensive, significantly higher performance can be achieved through the use of faster hardware. Future indexing capabilities will also vastly increase overall performance when dealing with very large databases.

Back to Top

10. What are the machine resource requirements (Memory, disk space) for SNAPI?

Because SNAPI relies on the developer to supply data (the names to be compared), the disk space needed is negligible. Memory requirements depend on the use of the API. For example, an application that creates a name object for each name in its database and keeps all of these objects in memory will require much more memory than an application that re-creates name objects for each query.

Back to Top

Developer Questions

1. Which header files should I include in my programs?

programs that use SNAPI classes or functions should include the *snapi.h* header file:

```
#include <snapi.h>
```

This file contains all necessary definitions for full use of the API.

Back to Top

2. Does the API support multi-threading?

SNAPI does not currently support multi-threading. This means that a multithreaded application should not allow two separate threads to access the same SNAPI objects at the same time. However, an application can perform two concurrent queries, each in a separate thread, so long as each thread has its own name objects (SNQueryNameData, SNEvalNameData) and results list object (SNResultsList).

Back to Top

3. How can I only look at surnames when doing a name comparison?

SNAPI provides parameters to weight the given name and surname fields relative to each other. This weight is applied when calculating a composite score for the name as a whole (the composite name score is calculated by performing a weighted average of the separate given name and surname scores). By setting the given name weight to 0.0, a developer can cause name comparisons to be based solely upon the surname. In doing so, the given name threshold should probably also be set to 0.0, to ensure that a poorly matched given name does not prevent a name from being considered a match. See documentation on the following functions for more details: setGnWeight(), setSnWeight(), setGnScoreThresh(), setSnScoreThresh().

Back to Top

4. I have a middle name field in my database. How do I include this information in my name comparisons?

SNAPI uses an internal name model that considers just given name and surname. However, both name objects (SNEvalNameData and SNQueryNameData) include a constructor that accepts a separate middle name. Internally, these constructors append this middle name to the given name.

Future versions may add middle name as an additional name field.

Back to Top

5. My application is written in (COBOL, FORTRAN, Java). How can I use SNAPI to add name checking to my app?

Direct use of the SNAPI requires a C++ compiler. Many programming environments include the ability to link in object code (compiled code) from other languages. For example, a mainframe COBOL application could call code written by a developer in C++

that uses SNAPI.

Another option is to implement a simple name check server that runs in a separate process. This simple server is written in C++, and accepts name check requests from client applications (e.g. a Java application). The method of communication between the client application and the server is up to the developer (A common method is to use a TCP-IP based socket).

Implementation of a server offers additional benefits. It provides a modular solution that can easily be upgraded and worked on separate from the client application. In addition, the server can then be run on a separate machine, allowing the hardware to be scaled independently for the client and server.

Back to Top

6. How do I associate other data (such as database record ids) with the name objects SNAPI uses? I need this so that when SNAPI tells me a name matches my query, I can look up other information associated with that name.

The SNAPI name objects (SNEvalNameData and SNQueryNameData) only include attributes for name information. However, the developer can create subclasses of these objects so that addition application specific data can be associated with each name object.

For example, a developer can create a subclass of SNEvalNameData called MySNEvalNameData, and add a new attribute called dbRecId. The constructor for this new class includes an extra parameter for a database record id (dbRecId), so that the value of this parameter can be set. Additionally, the class includes a method to return the value of the parameter. During a search, each time a name is retrieved from the database, a new MySNEvalNameData object is constructed along with its associated dbRecId. After the query is completed, the objects in the results list can then be inspected to determine their associated database record id.

See the subclassing sections of SNEvalNameData and SNQuervNameData for more details.

Back to Top

7. I have other information (such as age and social security number) that I want to include in the comparison process. How do I do this?

By subclassing the SNAPI name classes (SNEvalNameData and SNQuervNameData), a developer can add new data elements into the comparison processes.

The SNEvalNameData class contains methods to perform score calculations and match determination. These methods are virtual, so that a developer can alter or extend the functionality to include data specific to their application.

For example, an SSN data element could be added to subclasses of SNEvalNameData and SNQueryNameData. The subclassed SNEvalNameData class could then override any or all of the calcComponentScores(), calcNameScore(), compareScore(), getCompResult() or resetScores() methods to integrate the SSN value. One might choose to have the calcComponentScores() method perform a comparison between the query name's SSN and the evaluation name's SSN. If the two matched exactly, the getCompResult() method could be altered to relax the name score threshold if the SSN values matched exactly.

This is just one possible implementation. See the subclassing sections of SNEvalNameData and SNQueryNameData for more details.

Back to Top

8. Which C++ compilers does SNAPI support?

SNAPI can be compiled with any modern C++ compiler. LAS has developed sample applications using Microsoft Visual C++ 4.2 and 5.0 compilers, as well the GNU g++ compiler under Solaris 2.5 and Linux 2.0.

Back to Top

9. My database contains names of mixed culture. Which culture should I specify when I create my parameters object?

The Anglo culture is generally suitable for use in "generic" queries. This covers queries where the culture of the names are Anglo, unknown or mixed. In addition, a developer may either allow users to specify a culture when performing a query or establish pre-defined cultural searches. The user-specified or pre-defined culture could then be used to create all requisite objects for performing a comparison. this approach would require the developer to create a new SNQueryParms object for each query, as well as ensure that the SNEvalNameData and SNQueryNameData objects are created using the same SNQueryParms object.

Future versions may include the ability to automatically classify the culture of a name and set search parameters as appropriate.

Back to Top

10. My database has over 10 million names. Can SNAPI handle this kind of volume?

The current version of SNAPI is capable of performing approximately 10,000 name comparisons per second on an Intel Pentium 133 MHz machine. Because name comparisons are CPU intensive, faster hardware can dramatically increase search times. There are also other mechanisms that can be used to address performance issues:

- Developers can often find ways of segmenting their data to avoid having to search the entire database for each query.
- Name objects can be constructed and stored in memory so that they do not have to be retrieved and constructed for each query. This can require large amounts of memory.
- Indexing capabilities will be available in the next version of the API.

Back to Top

11. I am planning to create name objects for every name in my database, and hold them in memory, reusing them for each query. How much memory will I need to do this?

The size of an SNEvalNameData object depends on the length of the name it represents. An average name requires approximately 600 bytes (the actual amount of storage can vary depending on the compiler used). Thus a database of 30,000 names would require about 18 - 20 megs to be stored in memory.

Back to Top

12. How can I change the scores assigned to a particular name variant association. How can I delete variant associations or add new ones?

The current version of SNAPI does permit the developer to make modifications to the variant information. In addition, variant checking can be turned off entirely. See the setUseGnVariants() and setUseSnVariants() methods for details.

Back to Top

13. How can I delete TAQ values or add new ones?

The current version of SNAPI does permit the developer to make modifications to the TAQ information. In addition, the way TAQs are processed can be adjusted by the developer. See the setGnTAQProcessingMode() and setSnTAQProcessingMode() methods for more details.

Back to Top

14. What types of name formats does SNAPI support?

The developer is responsible for accessing application data (e.g. via calls to a database or reading from a file). Therefore, SNAPI does not care about how the data is stored. However, SNAPI does provide several ways of constructing name objects:

1. Given name and surname specified as separate string variables.
2. Given name, middle name, and surname specified as separate string variables.
3. Name specified as a single string in a comma delimited format (surname, given name).
4. Name specified as a single string, with the last stem segment interpreted as the surname.

The first form is the most efficient, because no parsing has to be done to separate the name into given name and surname. The fourth form includes advanced processing to identify TAQ values and exclude them when determining the surname.

Back to Top

11. I am planning to create name objects for every name in my database, and hold them in memory, reusing them for each query. How much memory will I need to do this?

The size of an SNEvalNameData object depends on the length of the name it represents. An average name requires approximately 600 bytes (the actual amount of storage can vary depending on the compiler used). Thus a database of 30,000 names would require about 18 - 20 megs to be stored in memory.

Back to Top

---

12. How can I change the scores assigned to a particular name variant association. How can I delete variant associations or add new ones?

The current version of SNAPI does permit the developer to make modifications to the variant information. In addition, variant checking can be turned off entirely. See the setUseGnVariants() and setUseSnVariants() methods for details.

Back to Top

---

13. How can I delete TAQ values or add new ones?

The current version of SNAPI does permit the developer to make modifications to the TAQ information. In addition, the way TAQs are processed can be adjusted by the developer. See the setGnTAQProcessingMode() and setSnTAQProcessingMode() methods for more details.

Back to Top

---

14. What types of name formats does SNAPI support?

The developer is responsible for accessing application data (e.g. via calls to a database or reading from a file). Therefore, SNAPI does not care about how the data is stored. However, SNAPI does provide several ways of constructing name objects:

1. Given name and surname specified as separate string variables.
2. Given name, middle name, and surname specified as separate string variables.
3. Name specified as a single string in a comma delimited format (surname, given name).
4. Name specified as a single string, with the last stem segment interpreted as the surname.

The first form is the most efficient, because no parsing has to be done to separate the name into given name and surname. The fourth form includes advanced processing to identify TAQ values and exclude them when determining the surname.

Back to Top

---

11. I am planning to create name objects for every name in my database, and hold them in memory, reusing them for each query. How much memory will I need to do this?

The size of an SNEvalNameData object depends on the length of the name it represents. An average name requires approximately 600 bytes (the actual amount of storage can vary depending on the compiler used). Thus a database of 30,000 names would require about 18 - 20 megs to be stored in memory.

Back to Top

---

12. How can I change the scores assigned to a particular name variant association. How can I delete variant associations or add new ones?

The current version of SNAPI does permit the developer to make modifications to the variant information. In addition, variant checking can be turned off entirely. See the setUseGnVariants() and setUseSnVariants() methods for details.

Back to Top

---

13. How can I delete TAQ values or add new ones?

The current version of SNAPI does permit the developer to make modifications to the TAQ information. In addition, the way TAQs are processed can be adjusted by the developer. See the setGnTAQProcessingMode() and setSnTAQProcessingMode() methods for more details.

Back to Top

---

14. What types of name formats does SNAPI support?

The developer is responsible for accessing application data (e.g. via calls to a database or reading from a file). Therefore, SNAPI does not care about how the data is stored. However, SNAPI does provide several ways of constructing name objects:

1. Given name and surname specified as separate string variables.
2. Given name, middle name, and surname specified as separate string variables.
3. Name specified as a single string in a comma delimited format (surname, given name).
4. Name specified as a single string, with the last stem segment interpreted as the surname.

The first form is the most efficient, because no parsing has to be done to separate the name into given name and surname. The fourth form includes advanced processing to identify TAQ values and exclude them when determining the surname.

Back to Top

Questions or problems regarding this web site should be directed to webmaster@las-inc.com.

Last modified: Friday January 23, 1998.

NAMEHUNTER

```
//      File:  NH_util.cpp
//
//      Description:
//
//              Implementation of various utility functions used in the SNAPI
//
//
//      History:
//
//              5/15/97         EFB             Created
//              3/20/98         EFB             Changed names to NH from SN
//


#include        <string.h>


#include        "NH_util.hpp"
#include        "NHCompParms.hpp"




//      function to remove leading and trailing spaces from a string
//      in place.
// Strips the string at either end or both ends.
// Stripchars specify the characters that should
// be stripped.  We start by seeing if they want the
// trailing chars stripped, which is easy.  We simply
// work backwards from the end of the string, looking for
// the first non-strippable character, and terminate the
// string just past that character.  Then if they wanted
// leading chars stripped, we work forwards to the first
// non-strippable char, and then move that and each following
// char to the beginning of the string.
void    NH_strip(char *aString)
{
 char *end_point;
 char *ch;
 int  len;

 if ((len = strlen(aString)) != 0) { // if there is a string
                // start at end
                end_point = aString + len - 1;
```

```
        // and work back till we get a non-space or get to
        // the begining of our string, chopping off what's left.
        // Also make sure we don't zoom right past the beginning of the
        // string.
        for (; strchr(NH_DEFAULT_WHITESPACE, *end_point) != NULL &&
end_point != aString; end_point--)
            ;
        // if string was all whitespace
        if ((end_point == aString) && strchr(NH_DEFAULT_WHITESPACE,
*aString) != NULL)
            *aString = EOS;  // erase it all, and we're done, could return here
        else
            *(end_point + 1) = EOS;   // just chop off excess blanks

        // make sure there is still a string, since it might
        // have been stripped entirely above.
        if (*aString) {
            // now find first non space.  we know string has at least one
            // nonwhite space, so we don't have to check for NULL.
            for (ch = aString; strchr(NH_DEFAULT_WHITESPACE, *ch) !=
NULL; ch++)
                ;
            if (ch != aString) { // if there were leading spaces, move the block
back
                char  *target = aString;
                while (*ch != EOS) {
                    *target = *ch;
                    target++;
                    ch++;
                }
                // and get the null char also
                *target = *ch;
            } // end if (are there leading spaces?)
        } // end if (and text left?)
  } // end (is there a string at all ?)
}


char    *       NH_strrchr(char *stringStart, char *searchPos, char searchChar)
{
    while (1)      {
        if (*searchPos == searchChar)
            break;
        if (searchPos == stringStart)  {
```

```
                        searchPos = NULL;           //      string not found, so return
NULL
                        break;
                    }
                    searchPos--;
                }
                return searchPos;
        }
```

```
//
//      File:  NH_queens_arrays.hpp
//
//      Description:
//
//              Contains global definitions and declarations for the valid
//              combinations of indexes for the best score calculation
//
//
//      History:
//
//              6/4/97  EFB             Created
//              3/20/98         EFB             Changed names to NH from SN
//

typedef         unsigned char byte;


        byte    twoByTwo[] = {1, 0,
                                                                0, 1};

        byte    twoByThree[] = {        1, 2,

        1, 0,

        2, 1,

        2, 0,

        0, 1,

        0, 2};

        byte    twoByFour[] = {         1, 2,

        1, 3,

        1, 0,

        2, 1,

        2, 3,

        2, 0,
3, 1,
3, 2,
3, 0,
0, 1,
0, 2,
0, 3};
byte    twoByFive[] = {       1, 2,
1, 3,
1, 4,
1, 0,
2, 1,
2, 3,
2, 4,
2, 0,
3, 1,
3, 2,
3, 4,
3, 0,
4, 1,
4, 2,
4, 3,
4, 0,
0, 1,
0, 2,
0, 3,
0, 4};
byte    threeByThree[] = {    1, 2, 0,
1, 0, 2,
2, 1, 0,
2, 0, 1,
0, 1, 2,
0, 2, 1};
byte    threeByFour[] = {     1, 2, 3,
1, 2, 0,
1, 3, 2,
1, 3, 0,
1, 0, 2,
1, 0, 3,
2, 1, 3,
2, 1, 0,
2, 3, 1,
2, 3, 0,
2, 0, 1,
2, 0, 3,
3, 1, 2,
3, 1, 0,
3, 2, 1,
3, 2, 0,
3, 0, 1,
3, 0, 2,
0, 1, 2,
0, 1, 3,
0, 2, 1,
0, 2, 3,
0, 3, 1,
0, 3, 2};
byte    threeByFive[] = {     1, 2, 3,
1, 2, 4,
1, 2, 0,
1, 3, 2,
1, 3, 4,
1, 3, 0,
1, 4, 2,
1, 4, 3,
1, 4, 0,
1, 0, 2,
```

1, 0, 3,

1, 0, 4,

2, 1, 3,

2, 1, 4,

2, 1, 0,

2, 3, 1,

2, 3, 4,

2, 3, 0,

2, 4, 1,

2, 4, 3,

2, 4, 0,

2, 0, 1,

2, 0, 3,

2, 0, 4,

3, 1, 2,

3, 1, 4,

3, 1, 0,

3, 2, 1,

3, 2, 4,

3, 2, 0,

3, 4, 1,

3, 4, 2,

3, 4, 0,

3, 0, 1,

3, 0, 2,

3, 0, 4,

4, 1, 2,

4, 1, 3,

4, 1, 0,

4, 2, 1,

4, 2, 3,

4, 2, 0,

4, 3, 1,

4, 3, 2,

4, 3, 0,

4, 0, 1,

4, 0, 2,

4, 0, 3,

0, 1, 2,

0, 1, 3,

0, 1, 4,

0, 2, 1,

0, 2, 3,

0, 2, 4,

```
0, 3, 1,

0, 3, 2,

0, 3, 4,

0, 4, 1,

0, 4, 2,

0, 4, 3};

byte     fourByFour[] =         {         1, 2, 3, 0,

1, 2, 0, 3,

1, 3, 0, 2,

1, 3, 2, 0,

1, 0, 2, 3,

1, 0, 3, 2,

2, 1, 3, 0,

2, 1, 0, 3,

2, 3, 1, 0,

2, 3, 0, 1,

2, 0, 1, 3,

2, 0, 3, 1,

3, 1, 2, 0,

3, 1, 0, 2,

3, 2, 1, 0,

3, 2, 0, 1,
3, 0, 1, 2,
3, 0, 2, 1,
0, 1, 2, 3,
0, 1, 3, 2,
0, 2, 1, 3,
0, 2, 3, 1,
0, 3, 1, 2,
0, 3, 2, 1};
byte    fourByFive[] =        {        1, 2, 3, 4,
1, 2, 3, 0,
1, 2, 4, 3,
1, 2, 4, 0,
1, 2, 0, 3,
1, 2, 0, 4,
1, 3, 2, 4,
1, 3, 2, 0,
1, 3, 4, 2,
1, 3, 4, 0,
1, 3, 0, 2,
1, 3, 0, 4,
1, 4, 2, 3,
1, 4, 2, 0,
```

1, 4, 3, 2,

1, 4, 3, 0,

1, 4, 0, 2,

1, 4, 0, 3,

1, 0, 2, 3,

1, 0, 2, 4,

1, 0, 3, 2,

1, 0, 3, 4,

1, 0, 4, 2,

1, 0, 4, 3,

2, 1, 3, 4,

2, 1, 3, 0,

2, 1, 4, 3,

2, 1, 4, 0,

2, 1, 0, 3,

2, 1, 0, 4,

2, 3, 1, 4,

2, 3, 1, 0,

2, 3, 4, 1,

2, 3, 4, 0,

2, 3, 0, 1,

2, 3, 0, 4,

2, 4, 1, 3,

2, 4, 1, 0,

2, 4, 3, 1,

2, 4, 3, 0,

2, 4, 0, 1,

2, 4, 0, 3,

2, 0, 1, 3,

2, 0, 1, 4,

2, 0, 3, 1,

2, 0, 3, 4,

2, 0, 4, 1,

2, 0, 4, 3,

3, 2, 1, 4,

3, 2, 1, 0,

3, 2, 4, 1,

3, 2, 4, 0,

3, 2, 0, 1,

3, 2, 0, 4,

3, 1, 2, 4,

3, 1, 2, 0,

3, 1, 4, 2,

3, 1, 4, 0,

3, 1, 0, 2,

3, 1, 0, 4,

3, 4, 2, 1,

3, 4, 2, 0,

3, 4, 1, 2,

3, 4, 1, 0,

3, 4, 0, 2,

3, 4, 0, 1,

3, 0, 2, 1,

3, 0, 2, 4,

3, 0, 1, 2,

3, 0, 1, 4,

3, 0, 4, 2,

3, 0, 4, 1,

4, 2, 3, 1,

4, 2, 3, 0,

4, 2, 1, 3,

4, 2, 1, 0,

4, 2, 0, 3,

4, 2, 0, 1,

4, 3, 2, 1,

4, 3, 2, 0,

4, 3, 1, 2,

4, 3, 1, 0,

4, 3, 0, 2,

4, 3, 0, 1,

4, 1, 2, 3,

4, 1, 2, 0,

4, 1, 3, 2,

4, 1, 3, 0,

4, 1, 0, 2,

4, 1, 0, 3,

4, 0, 2, 3,

4, 0, 2, 1,

4, 0, 3, 2,

4, 0, 3, 1,

4, 0, 1, 2,

4, 0, 1, 3,

0, 2, 3, 4,

0, 2, 3, 1,

0, 2, 4, 3,

0, 2, 4, 1,

0, 2, 1, 3,

0, 2, 1, 4,

```
0, 3, 2, 4,
0, 3, 2, 1,
0, 3, 4, 2,
0, 3, 4, 1,
0, 3, 1, 2,
0, 3, 1, 4,
0, 4, 2, 3,
0, 4, 2, 1,
0, 4, 3, 2,
0, 4, 3, 1,
0, 4, 1, 2,
0, 4, 1, 3,
0, 1, 2, 3,
0, 1, 2, 4,
0, 1, 3, 2,
0, 1, 3, 4,
0, 1, 4, 2,
0, 1, 4, 3};

byte    fiveByFive[] ={         1, 2, 3, 4, 0,
1, 2, 3, 0, 4,
1, 2, 4, 3, 0,
1, 2, 4, 0, 3,
```

1, 2, 0, 3, 4,

1, 2, 0, 4, 2,

1, 3, 2, 4, 0,

1, 3, 2, 0, 4,

1, 3, 4, 2, 0,

1, 3, 4, 0, 2,

1, 3, 0, 2, 4,

1, 3, 0, 4, 2,

1, 4, 2, 3, 0,

1, 4, 2, 0, 3,

1, 4, 3, 2, 0,

1, 4, 3, 0, 2,

1, 4, 0, 2, 3,

1, 4, 0, 3, 2,

1, 0, 2, 3, 4,

1, 0, 2, 4, 3,

1, 0, 3, 2, 4,

1, 0, 3, 4, 2,

1, 0, 4, 2, 3,

1, 0, 4, 3, 2,

2, 1, 3, 4, 0,

2, 1, 3, 0, 4,

2, 1, 4, 3, 0,

2, 1, 4, 0, 3,

2, 1, 0, 3, 4,

2, 1, 0, 4, 1,

2, 3, 1, 4, 0,

2, 3, 1, 0, 4,

2, 3, 4, 1, 0,

2, 3, 4, 0, 1,

2, 3, 0, 1, 4,

2, 3, 0, 4, 1,

2, 4, 1, 3, 0,

2, 4, 1, 0, 3,

2, 4, 3, 1, 0,

2, 4, 3, 0, 1,

2, 4, 0, 1, 3,

2, 4, 0, 3, 1,

2, 0, 1, 3, 4,

2, 0, 1, 4, 3,

2, 0, 3, 1, 4,

2, 0, 3, 4, 1,

2, 0, 4, 1, 3,

2, 0, 4, 3, 1,

3, 2, 1, 4, 0,

3, 2, 1, 0, 4,

3, 2, 4, 1, 0,

3, 2, 4, 0, 1,

3, 2, 0, 1, 4,

3, 2, 0, 4, 2,

3, 1, 2, 4, 0,

3, 1, 2, 0, 4,

3, 1, 4, 2, 0,

3, 1, 4, 0, 2,

3, 1, 0, 2, 4,

3, 1, 0, 4, 2,

3, 4, 2, 1, 0,

3, 4, 2, 0, 1,

3, 4, 1, 2, 0,

3, 4, 1, 0, 2,

3, 4, 0, 2, 1,

3, 4, 0, 1, 2,

3, 0, 2, 1, 4,

3, 0, 2, 4, 1,

3, 0, 1, 2, 4,

3, 0, 1, 4, 2,

3, 0, 4, 2, 1,

3, 0, 4, 1, 2,

4, 2, 3, 1, 0,

4, 2, 3, 0, 1,

4, 2, 1, 3, 0,

4, 2, 1, 0, 3,

4, 2, 0, 3, 1,

4, 2, 0, 1, 2,

4, 3, 2, 1, 0,

4, 3, 2, 0, 1,

4, 3, 1, 2, 0,

4, 3, 1, 0, 2,

4, 3, 0, 2, 1,

4, 3, 0, 1, 2,

4, 1, 2, 3, 0,

4, 1, 2, 0, 3,

4, 1, 3, 2, 0,

4, 1, 3, 0, 2,

4, 1, 0, 2, 3,

4, 1, 0, 3, 2,

4, 0, 2, 3, 1,

4, 0, 2, 1, 3,

4, 0, 3, 2, 1,

4, 0, 3, 1, 2,

4, 0, 1, 2, 3,

4, 0, 1, 3, 2,

0, 2, 3, 4, 1,

0, 2, 3, 1, 4,

0, 2, 4, 3, 1,

0, 2, 4, 1, 3,

0, 2, 1, 3, 4,

0, 2, 1, 4, 2,

0, 3, 2, 4, 1,

0, 3, 2, 1, 4,

0, 3, 4, 2, 1,

0, 3, 4, 1, 2,

0, 3, 1, 2, 4,

0, 3, 1, 4, 2,

0, 4, 2, 3, 1,

0, 4, 2, 1, 3,

0, 4, 3, 2, 1,

0, 4, 3, 1, 2,

0, 4, 1, 2, 3,

0, 4, 1, 3, 2,

```
0, 1, 2, 3, 4,

0, 1, 2, 4, 3,

0, 1, 3, 2, 4,

0, 1, 3, 4, 2,

0, 1, 4, 2, 3,

0, 1, 4, 3, 2};
```

```
4, 3, 1, 0,
4, 3, 0, 2,
4, 3, 0, 1,
4, 1, 2, 3,
4, 1, 2, 0,
4, 1, 3, 2,
4, 1, 3, 0,
4, 1, 0, 2,
4, 1, 0, 3,
4, 0, 2, 3,
4, 0, 2, 1,
4, 0, 3, 2,
4, 0, 3, 1,
4, 0, 1, 2,
4, 0, 1, 3,
0, 2, 3, 4,
0, 2, 3, 1,
0, 2, 4, 3,
0, 2, 4, 1,
0, 2, 1, 3,
0, 2, 1, 4,
0, 3, 2, 4,
0, 3, 2, 1,
0, 3, 4, 2,
0, 3, 4, 1,
0, 3, 1, 2,
0, 3, 1, 4,
0, 4, 2, 3,
0, 4, 2, 1,
0, 4, 3, 2,
0, 4, 3, 1,
0, 4, 1, 2,
0, 4, 1, 3,
0, 1, 2, 3,
0, 1, 2, 4,
0, 1, 3, 2,
0, 1, 3, 4,
0, 1, 4, 2,
0, 1, 4, 3};
byte  fiveByFive[] =     {      1, 2, 3, 4, 0,
1, 2, 3, 0, 4,
1, 2, 4, 3, 0,
1, 2, 4, 0, 3,
1, 2, 0, 3, 4,
1, 2, 0, 4, 2,
1, 3, 2, 4, 0,
1, 3, 2, 0, 4,
1, 3, 4, 2, 0,
1, 3, 4, 0, 2,
1, 3, 0, 2, 4,
1, 3, 0, 4, 2,
1, 4, 2, 3, 0,
1, 4, 2, 0, 3,
1, 4, 3, 2, 0,
1, 4, 3, 0, 2,
1, 4, 0, 2, 3,
1, 4, 0, 3, 2,
1, 0, 2, 3, 4,
1, 0, 2, 4, 3,
1, 0, 3, 2, 4,
1, 0, 3, 4, 2,
1, 0, 4, 2, 3,
1, 0, 4, 3, 2,
2, 1, 3, 4, 0,
2, 1, 3, 0, 4,
2, 1, 4, 3, 0,
2, 1, 4, 0, 3,
2, 1, 0, 3, 4,
2, 1, 0, 4, 1,
2, 3, 1, 4, 0,
2, 3, 1, 0, 4,
2, 3, 4, 1, 0,
2, 3, 4, 0, 1,
2, 3, 0, 1, 4,
2, 3, 0, 4, 1,
2, 4, 1, 3, 0,
2, 4, 1, 0, 3,
2, 4, 3, 1, 0,
2, 4, 3, 0, 1,
2, 4, 0, 1, 3,
2, 4, 0, 3, 1,
2, 0, 1, 3, 4,
2, 0, 1, 4, 3,
2, 0, 3, 1, 4,
2, 0, 3, 4, 1,
2, 0, 4, 1, 3,
2, 0, 4, 3, 1,
3, 2, 1, 4, 0,
3, 2, 1, 0, 4,
3, 2, 4, 1, 0,
3, 2, 4, 0, 1,
3, 2, 0, 1, 4,
3, 2, 0, 4, 2,
3, 1, 2, 4, 0,
3, 1, 2, 0, 4,
3, 1, 4, 2, 0,
3, 1, 4, 0, 2,
3, 1, 0, 2, 4,
3, 1, 0, 4, 2,
3, 4, 2, 1, 0,
3, 4, 2, 0, 1,
3, 4, 1, 2, 0,
3, 4, 1, 0, 2,
3, 4, 0, 2, 1,
3, 4, 0, 1, 2,
3, 0, 2, 1, 4,
3, 0, 2, 4, 1,
3, 0, 1, 2, 4,
3, 0, 1, 4, 2,
3, 0, 4, 2, 1,
3, 0, 4, 1, 2,
4, 2, 3, 1, 0,
4, 2, 3, 0, 1,
4, 2, 1, 3, 0,
4, 2, 1, 0, 3,
4, 2, 0, 3, 1,
4, 2, 0, 1, 2,
4, 3, 2, 1, 0,
4, 3, 2, 0, 1,
4, 3, 1, 2, 0,
4, 3, 1, 0, 2,
4, 3, 0, 2, 1,
4, 3, 0, 1, 2,
4, 1, 2, 3, 0,
4, 1, 2, 0, 3,
4, 1, 3, 2, 0,
4, 1, 3, 0, 2,
4, 1, 0, 2, 3,
4, 1, 0, 3, 2,
4, 0, 2, 3, 1,
4, 0, 2, 1, 3,
4, 0, 3, 2, 1,
4, 0, 3, 1, 2,
4, 0, 1, 2, 3,
4, 0, 1, 3, 2,
0, 2, 3, 4, 1,
0, 2, 3, 1, 4,
0, 2, 4, 3, 1,
0, 2, 4, 1, 3,
0, 2, 1, 3, 4,
0, 2, 1, 4, 2,
0, 3, 2, 4, 1,
0, 3, 2, 1, 4,
0, 3, 4, 2, 1,
0, 3, 4, 1, 2,
0, 3, 1, 2, 4,
0, 3, 1, 4, 2,
0, 4, 2, 3, 1,
0, 4, 2, 1, 3,
0, 4, 3, 2, 1,
0, 4, 3, 1, 2,
0, 4, 1, 2, 3,
0, 4, 1, 3, 2,
0, 1, 2, 3, 4,
0, 1, 2, 4, 3,
0, 1, 3, 2, 4,
0, 1, 3, 4, 2,
0, 1, 4, 2, 3,
0, 1, 4, 3, 2};
```

```
//     File:  NH_getErrorText.cpp
//
//     Description:
//
//             Implementation to the NH_getErrorText function.  This
function can
//             be used to return the error text for an associated error
code.
//
//
//     History:
//
//             6/23/97     EFB         Created
//             3/20/98     EFB         Changed names to NH from SN
//


#include    "NH_get_error_text.h"


#include    <string.h>


void  NH_get_error_text(NHReturnCode errorCode, char *textBuffer, int
maxChars)
{
      char  *errorMsgPtr;

      switch       (errorCode) {
            case  NH_SUCCESS:
                  errorMsgPtr = "Operation successful";
                  break;
            case  NH_MATCH:
                  errorMsgPtr = "The comparison matched";
                  break;
            case  NH_NO_MATCH:
                  errorMsgPtr = "The comparison did not match";
                  break;
            case  NH_INVALID_SCORE_THRESH:
                  errorMsgPtr = "The threshold must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_GN_INIT_SCORE:
                  errorMsgPtr = "The GN initial score must be between
0.0 and 1.0";
                  break;
            case  NH_INVALID_NH_INIT_SCORE:
                  errorMsgPtr = "The SN initial score must be between
0.0 and 1.0";
                  break;
            case  NH_INVALID_GN_INIT_ON_INIT_MATCH_SCORE:
                  errorMsgPtr = "The GN initial on intial match score
must be between 0.0 and 1.0";
                  break;
            case  NH_INVALID_NH_INIT_ON_INIT_MATCH_SCORE:
                  errorMsgPtr = "The SN initial on intial match score
must be between 0.0 and 1.0";
                  break;
            case  NH_INVALID_NFN_SCORE:
                  errorMsgPtr = "The NFN score must be between 0.0 and
1.0";
```

```
                  break;
            case  NH_INVALID_FNU_SCORE:
                  errorMsgPtr = "The FNU score must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_NLN_SCORE:
                  errorMsgPtr = "The NLN score must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_LNU_SCORE:
                  errorMsgPtr = "The LNU score must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_GN_ANCHOR_FACTOR:
                  errorMsgPtr = "The GN anchor score must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_NH_ANCHOR_FACTOR:
                  errorMsgPtr = "The SN anchor score must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_GN_OOPS_FACTOR:
                  errorMsgPtr = "The GN OOPS factor must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_NH_OOPS_FACTOR:
                  errorMsgPtr = "The SN OOPS factor must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_ABS_DEL_GN_TAQ_FACTOR:
                  errorMsgPtr = "The Abs delete GN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_ABS_DIS_GN_TAQ_FACTOR:
                  errorMsgPtr = "The Abs disregard GN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_ABS_DEL_NH_TAQ_FACTOR:
                  errorMsgPtr = "The Abs delete SN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_ABS_DIS_NH_TAQ_FACTOR:
                  errorMsgPtr = "The Abs disregard SN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_DEL_GN_TAQ_FACTOR:
                  errorMsgPtr = "The delete GN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_DIS_GN_TAQ_FACTOR:
                  errorMsgPtr = "The disregard GN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_DEL_NH_TAQ_FACTOR:
                  errorMsgPtr = "The delete SN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_DIS_NH_TAQ_FACTOR:
                  errorMsgPtr = "The disregard SN TAQ factor must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_GN_COMPRESSED_NAME_SCORE:
                  errorMsgPtr = "The GN compressed name score must be
between 0.0 and 1.0";
                  break;
            case  NH_INVALID_NH_COMPRESSED_NAME_SCORE:
                  errorMsgPtr = "The SN compressed name score must be
between 0.0 and 1.0";
                  break;
            case  NH_RESULTS_LIST_INSERT_ALLOC_FAILURE:
                  errorMsgPtr = "Could not allocate space for a new
results list";
                  break;
            case  NH_GN_VAR_TABLE_CREATION_ERROR:
                  errorMsgPtr = "Problem creating GN variant table";
                  break;
            case  NH_NH_VAR_TABLE_CREATION_ERROR:
                  errorMsgPtr = "Problem creating SN variant table";
                  break;
            case  NH_TAQ_TABLE_CREATION_ERROR:
                  errorMsgPtr = "Problem creating TAQ table";
                  break;
            case  NH_SEG_BREAK_CHARS_CREATION_ERROR:
                  errorMsgPtr = "Problem creating segment break
characters string";
                  break;
            case  NH_NOISE_CHARS_CREATION_ERROR:
                  errorMsgPtr = "Problem creating noise characters
string";
                  break;
            case  NH_INVALID_RESULTS_LIST_SIZE:
                  errorMsgPtr = "Invalid size requested for results
list";
                  break;
            case  NH_RESULTS_LIST_ALLOCATION_ERROR:
                  errorMsgPtr = "Problem creating internal results list
storage";
                  break;
            case  NH_RESULTS_ARRAY_NULL_ERROR:
                  errorMsgPtr = "Internal results list storage is
invalid";
                  break;
            case  NH_TAQ_RECORD_ALLOC_ERROR:
                  errorMsgPtr = "Problem allocating space for new TAQ
record";
                  break;
            case  NH_VARIANT_ALLOC_ERROR:
                  errorMsgPtr = "Problem allocating space for new
variant record";
                  break;
            case  NH_VARIANTS_DONT_EXIST:
                  errorMsgPtr = "The supplied names are not currently
variants";
                  break;
            case  NH_INVALID_VARIANT_SCORE:
                  errorMsgPtr = "Variant scores must be between 0.0 and
1.0";
                  break;
            case  NH_MAX_VARIANT_SIZE_INCREMENT_FAILED:
                  errorMsgPtr = "Could not increase variant storage to
add new variant relationship";
                  break;
            case  NH_VARIANT_ALREADY_RELATED:
                  errorMsgPtr = "The names are already related to each
other";
                  break;
            case  NH_COMP_PARMS_BAD_STREAM_ON_CONSTRUCT:
                  errorMsgPtr = "The comp parameters stream passed to
the constructor is invalid";
                  break;
            case  NH_COMP_PARMS_BAD_STREAM_ON_ARCHIVE:
                  errorMsgPtr = "The comp parameters stream passed to
the archiveData method is invalid";
                  break;
            case  NH_NAME_PARMS_FILE_NOISE_CHARS_ERROR:
                  errorMsgPtr = "The noise characters could not be
read";
                  break;
            case  NH_NAME_PARMS_FILE_BREAKS_CHARS_ERROR:
                  errorMsgPtr = "The break characters could not be
read";
                  break;
            case  NH_NAME_PARMS_BAD_STREAM_ON_CONSTRUCT:
                  errorMsgPtr = "The Name Parameters stream passed to
the constructor was bad";
                  break;
            case  NH_NAME_PARMS_BAD_STREAM_ON_WRITE:
                  errorMsgPtr = "The Name Parameters stream passed to
the archive method was bad";
                  break;
            case  NH_NAME_PARMS_FILE_BAD_CULTURE_CODE:
                  errorMsgPtr = "The culture code read from the Name
parameters stream was invalid";
                  break;
            case  NH_TAQ_NOT_FOUND:
                  errorMsgPtr = "The specified TAQ could not be found";
                  break;
            case  NH_TAQ_ALREADY_EXISTS:
                  errorMsgPtr = "The specified TAQ is already defined";
                  break;
            case  NH_INVALID_GN_THRESH:
                  errorMsgPtr = "The GN Threshold must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_NH_THRESH:
                  errorMsgPtr = "The SN Threshold must be between 0.0
and 1.0";
                  break;
            case  NH_INVALID_GN_WEIGHT:
                  errorMsgPtr = "The GN Weight must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_NH_WEIGHT:
                  errorMsgPtr = "The SN Weight must be between 0.0 and
1.0";
                  break;
            case  NH_INVALID_CULTURE_CODE:
                  errorMsgPtr = "The specified culture code is invalid";
                  break;
            case  NH_ERROR_READING_CUSTOM_PARAMETER_FROM_FILE:
                  errorMsgPtr = "A problem was encounter when reading a
custom parameter from a file";
                  break;
            case  NH_ERROR_WRITING_CUSTOM_PARAMETER_TO_FILE:
                  errorMsgPtr = "A problem was encounter when writing a
custom parameter to a file";
                  break;
            default:
                  errorMsgPtr = "Unknown Error";
                  break;
      }
      strncpy(textBuffer, errorMsgPtr, maxChars);
      textBuffer[maxChars] = EOS;
}
```

```
//     File:  NH_culture_codes.cpp
//
//     Description:
//
//            Definition of global array of culture code strings
//
//
//     History:
//
//            9/12/97     EFB         Created
//            3/20/98     EFB         Changed names to NH from SN
//


#include     <string.h>

#include     "NH_culture_codes.h"


//     The following two global arrays must be the same size.
//     That is, they must have the same number of elements.
//     If you add or remove items, you must also update the
//     constant NH_NUM_CULTURE_CODES
//     In addition, they must maintain the same relative order
//     (for example, Arabic must be in the same position in both
//     arrays).
//     lastly, this stuff must match the NHParmsType enum type,
//     both in number and relative position. The NH_NUM_PARMS_TYPES
//     must also be kept in sync as well.
char   *NH_culture_codes[] = { NH_CULTURE_CODE_ANGLO,

                  NH_CULTURE_CODE_ARABIC,

                  NH_CULTURE_CODE_CHINESE,

                  NH_CULTURE_CODE_GENERIC,

                  NH_CULTURE_CODE_HISPANIC,

                  NH_CULTURE_CODE_KOREAN,

                  NH_CULTURE_CODE_RUSSIAN};

char   *NH_culture_strings[] = {     NH_CULTURE_STRING_ANGLO,

                  NH_CULTURE_STRING_ARABIC,

                  NH_CULTURE_STRING_CHINESE,

                  NH_CULTURE_STRING_GENERIC,

                  NH_CULTURE_STRING_HISPANIC,

                  NH_CULTURE_STRING_KOREAN,

                  NH_CULTURE_STRING_RUSSIAN};


bool   NH_validate_culture_code(NHCultureCode     cultureCode)
{
       bool   found = false;
```

```
      for (int i = 0; i < NH_NUM_CULTURE_CODES; i++)  {
            if (!strncmp(cultureCode, NH_culture_codes[i],
NH_MAX_CULTURE_CODE_LEN))     {
                  found = true;
                  break;
            }
      }
      return found;
}
```

```
//    File:  namehunter.h
//
//    Description:
//
//          shutdown and startup functions for the NameHunter system.
//          These are really just blind interfaces to the
//          NH_variant_taq_globals functions.  We do this because
//          we want to hide the details of the variants and TAQs
//          from the API user.
//
//
//    History:
//
//          9/9/97      EFB         Created
//          3/20/98     EFB         Changed names to NH from SN

#include    "namehunter.h"
#include    "NHVariantTable.hpp"
#include    "NHTAQTable.hpp"
#include    "NH_variant_taq_globals.h"
#include    "NHDigraphBitmapArray.hpp"


extern      NHVariantTable    *NH_snVariantTable;
extern      NHVariantTable    *NH_gnVariantTable;
extern      NHTAQTable              *NH_taqTable;


NHDigraphBitmapArray    globalDigraphBitmapArray;


void  NH_startup()
{
      NH_getVariantTable(NH_SURNAME_VARIANTS);
      NH_getVariantTable(NH_GIVENNAME_VARIANTS);
      NH_getTAQTable();
}


void  NH_shutdown()
{
      if (NH_snVariantTable != NULL)      {
            delete NH_snVariantTable;
            NH_snVariantTable = NULL;
      }
      if (NH_gnVariantTable != NULL)      {
            delete NH_gnVariantTable;
            NH_gnVariantTable = NULL;
      }
      if (NH_taqTable != NULL)      {
            delete NH_taqTable;
            NH_taqTable = NULL;
      }
}
```

```
//      File:  NH_getErrorText.cpp
//
//      Description:
//
//              Implementation to the NH_getErrorText function.  This function can
//              be used to return the error text for an associated error code.
//
//
//      History:
//
//              6/23/97     EFB         Created
//              3/20/98     EFB         Changed names to NH from SN
//


#include        "NH_get_error_text.h"


#include        <string.h>


void    NH_get_error_text(NHReturnCode errorCode, char *textBuffer, int maxChars)
{
        char    *errorMsgPtr;

        switch (errorCode)    {
                case   NH_SUCCESS:
                       errorMsgPtr = "Operation successful";
                       break;
                case   NH_MATCH:
                       errorMsgPtr = "The comparison matched";
                       break;
                case   NH_NO_MATCH:
                       errorMsgPtr = "The comparison did not match";
                       break;
                case   NH_INVALID_SCORE_THRESH:
                       errorMsgPtr = "The threshold must be between 0.0 and 1.0";
                       break;
                case   NH_INVALID_GN_INIT_SCORE:
                       errorMsgPtr = "The GN initial score must be between 0.0 and 1.0";
                       break;
                case   NH_INVALID_NH_INIT_SCORE:
                       errorMsgPtr = "The SN initial score must be between 0.0 and 1.0";
                       break;
                case   NH_INVALID_GN_INIT_ON_INIT_MATCH_SCORE:
                    errorMsgPtr = "The GN initial on intial match score must be
between 0.0 and 1.0";
                    break;
             case   NH_INVALID_NH_INIT_ON_INIT_MATCH_SCORE:
                    errorMsgPtr = "The SN initial on intial match score must be
between 0.0 and 1.0";
                    break;
             case   NH_INVALID_NFN_SCORE:
                    errorMsgPtr = "The NFN score must be between 0.0 and 1.0";
                    break;
             case   NH_INVALID_FNU_SCORE:
                    errorMsgPtr = "The FNU score must be between 0.0 and 1.0";
                    break;
             case   NH_INVALID_NLN_SCORE:
                    errorMsgPtr = "The NLN score must be between 0.0 and 1.0";
                    break;
             case   NH_INVALID_LNU_SCORE:
                    errorMsgPtr = "The LNU score must be between 0.0 and 1.0";
                    break;
             case   NH_INVALID_GN_ANCHOR_FACTOR:
                    errorMsgPtr = "The GN anchor score must be between 0.0 and
1.0";
                    break;
             case   NH_INVALID_NH_ANCHOR_FACTOR:
                    errorMsgPtr = "The SN anchor score must be between 0.0 and
1.0";
                    break;
             case   NH_INVALID_GN_OOPS_FACTOR:
                    errorMsgPtr = "The GN OOPS factor must be between 0.0 and
1.0";
                    break;
             case   NH_INVALID_NH_OOPS_FACTOR:
                    errorMsgPtr = "The SN OOPS factor must be between 0.0 and
1.0";
                    break;
             case   NH_INVALID_ABS_DEL_GN_TAQ_FACTOR:
                    errorMsgPtr = "The Abs delete GN TAQ factor must be between
0.0 and 1.0";
                    break;
             case   NH_INVALID_ABS_DIS_GN_TAQ_FACTOR:
                    errorMsgPtr = "The Abs disregard GN TAQ factor must be
between 0.0 and 1.0";
                    break;
             case   NH_INVALID_ABS_DEL_NH_TAQ_FACTOR:
```

```
                        errorMsgPtr = "The Abs delete SN TAQ factor must be between
0.0 and 1.0";
                        break;
                case    NH_INVALID_ABS_DIS_NH_TAQ_FACTOR:
                        errorMsgPtr = "The Abs disregard SN TAQ factor must be
between 0.0 and 1.0";
                        break;
                case    NH_INVALID_DEL_GN_TAQ_FACTOR:
                        errorMsgPtr = "The delete GN TAQ factor must be between 0.0
and 1.0";
                        break;
                case    NH_INVALID_DIS_GN_TAQ_FACTOR:
                        errorMsgPtr = "The disregard GN TAQ factor must be between 0.0
and 1.0";
                        break;
                case    NH_INVALID_DEL_NH_TAQ_FACTOR:
                        errorMsgPtr = "The delete SN TAQ factor must be between 0.0
and 1.0";
                        break;
                case    NH_INVALID_DIS_NH_TAQ_FACTOR:
                        errorMsgPtr = "The disregard SN TAQ factor must be between 0.0
and 1.0";
                        break;
                case    NH_INVALID_GN_COMPRESSED_NAME_SCORE:
                        errorMsgPtr = "The GN compressed name score must be between
0.0 and 1.0";
                        break;
                case    NH_INVALID_NH_COMPRESSED_NAME_SCORE:
                        errorMsgPtr = "The SN compressed name score must be between
0.0 and 1.0";
                        break;
                case    NH_RESULTS_LIST_INSERT_ALLOC_FAILURE:
                        errorMsgPtr = "Could not allocate space for a new results list";
                        break;
                case    NH_GN_VAR_TABLE_CREATION_ERROR:
                        errorMsgPtr = "Problem creating GN variant table";
                        break;
                case    NH_NH_VAR_TABLE_CREATION_ERROR:
                        errorMsgPtr = "Problem creating SN variant table";
                        break;
                case    NH_TAQ_TABLE_CREATION_ERROR:
                        errorMsgPtr = "Problem creating TAQ table";
                        break;
                case    NH_SEG_BREAK_CHARS_CREATION_ERROR:
                        errorMsgPtr = "Problem creating segment break characters string";
                        break;
                case    NH_NOISE_CHARS_CREATION_ERROR:
                        errorMsgPtr = "Problem creating noise characters string";
                        break;
                case    NH_INVALID_RESULTS_LIST_SIZE:
                        errorMsgPtr = "Invalid size requested for results list";
                        break;
                case    NH_RESULTS_LIST_ALLOCATION_ERROR:
                        errorMsgPtr = "Problem creating internal results list storage";
                        break;
                case    NH_RESULTS_ARRAY_NULL_ERROR:
                        errorMsgPtr = "Internal results list storage is invalid";
                        break;
                case    NH_TAQ_RECORD_ALLOC_ERROR:
                        errorMsgPtr = "Problem allocating space for new TAQ record";
                        break;
                case    NH_VARIANT_ALLOC_ERROR:
                        errorMsgPtr = "Problem allocating space for new variant record";
                        break;
                case    NH_VARIANTS_DONT_EXIST:
                        errorMsgPtr = "The supplied names are not currently variants";
                        break;
                case    NH_INVALID_VARIANT_SCORE:
                        errorMsgPtr = "Variant scores must be between 0.0 and 1.0";
                        break;
                case    NH_MAX_VARIANT_SIZE_INCREMENT_FAILED:
                        errorMsgPtr = "Could not increase variant storage to add new
variant relationship";
                        break;
                case    NH_VARIANT_ALREADY_RELATED:
                        errorMsgPtr = "The names are already related to each other";
                        break;
                case    NH_COMP_PARMS_BAD_STREAM_ON_CONSTRUCT:
                        errorMsgPtr = "The comp parameters stream passed to the
constructor is invalid";
                        break;
                case    NH_COMP_PARMS_BAD_STREAM_ON_ARCHIVE:
                        errorMsgPtr = "The comp parameters stream passed to the
archiveData method is invalid";
                        break;
                case    NH_NAME_PARMS_FILE_NOISE_CHARS_ERROR:
                        errorMsgPtr = "The noise characters could not be read";
                        break;
                case    NH_NAME_PARMS_FILE_BREAKS_CHARS_ERROR:
                        errorMsgPtr = "The break characters could not be read";
                        break;
                case    NH_NAME_PARMS_BAD_STREAM_ON_CONSTRUCT:
                        errorMsgPtr = "The Name Parameters stream passed to the
constructor was bad";
                        break;
                case    NH_NAME_PARMS_BAD_STREAM_ON_WRITE:
                        errorMsgPtr = "The Name Parameters stream passed to the archive
method was bad";
                        break;
                case    NH_NAME_PARMS_FILE_BAD_CULTURE_CODE:
                        errorMsgPtr = "The culture code read from the Name parameters
stream was invalid";
                        break;
                case    NH_TAQ_NOT_FOUND:
                        errorMsgPtr = "The specified TAQ could not be found";
                        break;
                case    NH_TAQ_ALREADY_EXISTS:
                        errorMsgPtr = "The specified TAQ is already defined";
                        break;
                case    NH_INVALID_GN_THRESH:
                        errorMsgPtr = "The GN Threshold must be between 0.0 and 1.0";
                        break;
                case    NH_INVALID_NH_THRESH:
                        errorMsgPtr = "The SN Threshold must be between 0.0 and 1.0";
                        break;
                case    NH_INVALID_GN_WEIGHT:
                        errorMsgPtr = "The GN Weight must be between 0.0 and 1.0";
                        break;
                case    NH_INVALID_NH_WEIGHT:
                        errorMsgPtr = "The SN Weight must be between 0.0 and 1.0";
                        break;
                case    NH_INVALID_CULTURE_CODE:
                        errorMsgPtr = "The specified culture code is invalid";
                        break;
                case
        NH_ERROR_READING_CUSTOM_PARAMETER_FROM_FILE:
                        errorMsgPtr = "A problem was encounter when reading a custom
parameter from a file";
                        break;
                case    NH_ERROR_WRITING_CUSTOM_PARAMETER_TO_FILE:
                        errorMsgPtr = "A problem was encounter when writing a custom
parameter to a file";
                        break;
                default:
                        errorMsgPtr = "Unknown Error";
```

```
            break;
    }
    strncpy(textBuffer, errorMsgPtr, maxChars);
    textBuffer[maxChars] = EOS;
}
```

```
//      File:  namehunter.h
//
//      Description:
//
//              shutdown and startup functions for the NameHunter system.
//              These are really just blind interfaces to the
//              NH_variant_taq_globals functions.  We do this because
//              we want to hide the details of the variants and TAQs
//              from the API user.
//
//
//      History:
//
//              9/9/97  EFB             Created
//              3/20/98         EFB             Changed names to NH from SN

#include        "namehunter.h"
#include        "NHVariantTable.hpp"
#include        "NHTAQTable.hpp"
#include        "NH_variant_taq_globals.h"
#include        "NHDigraphBitmapArray.hpp"


extern  NHVariantTable          *NH_snVariantTable;
extern  NHVariantTable          *NH_gnVariantTable;
extern  NHTAQTable                      *NH_taqTable;


NHDigraphBitmapArray            globalDigraphBitmapArray;


void    NH_startup()
{
        NH_getVariantTable(NH_SURNAME_VARIANTS);
        NH_getVariantTable(NH_GIVENNAME_VARIANTS);
        NH_getTAQTable();
}


void    NH_shutdown()
{
        if (NH_snVariantTable != NULL)     {
                delete NH_snVariantTable;
                NH_snVariantTable = NULL;
        }
        if (NH_gnVariantTable != NULL)     {
                delete NH_gnVariantTable;
                NH_gnVariantTable = NULL;
        }
        if (NH_taqTable != NULL)    {
                delete NH_taqTable;
                NH_taqTable = NULL;
        }
}
```

```
//    File:  NHVariantTable.hpp
//
//    Description:
//
//          Interface to the NHVariantTable class.
//
//
//    History:
//
//          5/7/97      EFB         Created
//          6/23/97     EFB         Changed processing to get rid of
variant types
//                                                    as assign an
individual score for each variant pair.
//          6/23/97     EFB         Enhanced comments
//          9/9/97      EFB         Added support for a culture code in
the variant object,
//                                                    which required
changes to this object's interaction
//                                                    with the NHVariant
class.
//          3/20/98     EFB         Changed names to NH from SN
//


/*

      Variant information consists of two names that are related, along
      with a designation of variant type, which describes how the two
      names are related.

      The following holds true in our model:

      -     if Name A is related to name B with varType V, then B is
            related to A with varType V.

      -     When constructing the table,
            only one of the pairs (A, B) or (B, A) should be entered.
The
            internals will ensure that a request of "is B related to A"
and
            a request of "is A related to B" will work.

      -     Name variants are single segments.

      Internally, we represent the information as a hash table of
      NH_VarHashTableRecord structures.  Each of these structures
      contains a name string, plus a Variant object.
      Each Variant object (a separate class) has the following:

      NHVarId                 id;
            //    unique id for each variant
      byte                    numRelatedVariants;     //    number of
other variants we are related to
      NHVarId                 variants[MAX_VARIANTS_PER_NAME]
//    array of id's
      double                  varScores[MAX_VARIANTS_PER_NAME]     //
score for each variant


//    as related to this variant
```

```
      short int         varCultures[MAX_VARIANTS_PER_NAME]  //    score
for each variant

//    as related to this variant

      The name of the variant is actually stored in the hash table node,
rather
      than the variant object.

      There are three important functions in the VariantTable class:


            bool              addVariant(char *name1, char *name2,
NHVarType varType, char *cultCode);
            NHVariant   getVariantObjectName(char *name);
            NHVarId           getVariantIdForName(char *name);


            //    The Variant has the method:
            double      getVariantScoreForIdAndCulture(NHVarId varId,
char *cultureCode);

      The variant table is built by multiple calls to addVariant() from
the
      constructor.  There is one call to addVariant() for each pair of
names
      that are related.

      addVariant() takes 2 names that are related, along with a culture
code to
      describe the relationship.

      getVariantInfoForName returns the NHVariant object associated with
the
      name (or NULL).

      getVariantIdForName() returns the id associated with the name.

      Typically, a QueryNameData object gets a pointer to it's variant
object
      up front.  Each time is gets compared to an EvalNameData object,
it
      calls the getVariantIdForName() method to get an id, which it then
passes
      the to the getVariantScoreForId() to see if the two are related.

*/


#ifndef     NHVARIANTTABLE_HPP
#define NHVARIANTTABLE_HPP


#include    "NHVariant.hpp"
#include    "NH_get_error_text.h"


//    define a const for end of string
#ifndef     EOS
```

```
#define     EOS   '\0'
#endif


//    how long can a variant be ?
#define           NH_MAX_VARIANT_LEN            30


//    define a type to specify the type of variant table
//    types are defined by a combination of culture and
//    name field.
enum  NH_VARIANT_TABLE_TYPES
{
      NH_SURNAME_VARIANTS,
      NH_GIVENNAME_VARIANTS,
      NH_EMPTY_VARIANTS
};


//    define a record in the Variant hash table
typedef struct NH_VAR_HASH_TABLE_RECORD_T {
      char
                        segment[NH_MAX_VARIANT_LEN + 1];
      NHVariant
                  *variant;
      struct
NH_VAR_HASH_TABLE_RECORD_T     *next;                  //    pointer to
next node in hash chain
} NH_VarHashTableRecord;

//    Do not change without seeing member function hash().
#define NH_MAX_VAR_HASH_TABLE_NODES 907

//    define a type that is a pointer to a NH_VarTableRecord
typedef NH_VarHashTableRecord *NH_VarHashTableRecordPtr;

//    define a type that is a table (array) of NH_VarTableRecord
typedef NH_VarHashTableRecordPtr
NH_VariantHashTable[NH_MAX_VAR_HASH_TABLE_NODES];


class NHVariantTable
{
      public:
            NHVariantTable(NH_VARIANT_TABLE_TYPES tableType);
            virtual ~NHVariantTable();


            //    returns the NHVariant object associated with the name,
            //    or NULL is there is no object for the name.
            NHVariant   *     getVariantObjectForName(char *name);

            //    returns the NHVarId associated with the name.  If
there is
            //    no variant for the name, the function returns
NH_VAR_NOT_FOUND.
            NHVarId           getVariantIdForName(char *name);

            NHReturnCode                        getStatus() {return
status;}
```

```
            NHReturnCode                        addVariant(char *name1,
char *name2, double varScore, char *cultCode);

            int         getNumHashBuckets()      {return
NH_MAX_VAR_HASH_TABLE_NODES;}

            NH_VarHashTableRecordPtr      getHashBucketStartNodeAt(int
hashTableIndex)

                        {return variantHashTable[hashTableIndex];}

            //    function to change the score associated with two
variants with a
            //    specified culture.
            //    The function return:
            //
            //          NH_SUCCESS - if things worked out OK
            //          NH_VARIANTS_DONT_EXIST - if the either name does
not exist in the table
            //
                              or the names are not already variants of
each
            //
                              other with the specified culture.
            //          NH_INVALID_VARIANT_SCORE - if the score is
invalid
            NHReturnCode      changeVariantScore(char *name1, char
*name2, char *cultureCode, double newScore);


            //    a function to remove the relationship between two
variants within
            //    a specified culture.
            //    This function is used for the VariantManager
application.
            //    If either variant ends up without a relationship after
this
            //    operation, it is left in, but when saved, the
resulting file
            //    will contain a "*" rather than a related name.  The
function can
            //    return
            //
            //          NH_SUCCESS - if things worked out OK
            //          NH_VARIANTS_DONT_EXIST - if the names are not
already variants
            NHReturnCode      removeVariantRelation(char *name1, char
*name2, char *cultureCode);

            //    return the next available id, which is the number of
            //    distinct variants in our table.
            NHVarId           getNextAvailableVarId() {return
nextAvailableVarId;}

            bool              getDirty()  {return dirty;}
            void              setDirty(bool aBool)    {dirty = aBool;}

      protected:

            //    add a variant relationship.
            virtual     NHVariant   *     getOrCreateVariantObjectForNam
```

```
e(char *name);

            NHVarId           nextAvailableVarId;


            NH_VariantHashTable           variantHashTable;

            NHReturnCode            status;                  //    are we
valid

            bool              dirty;             //    have we changed

            //    Returns an integer in the range [0,
NH_MAX_VAR_HASH_TABLE_NODES].
            inline unsigned int NHVariantTable::hash(char *string)
            {
                  char                                  *p;
                  unsigned    int          i;
                  unsigned    int          sum;

                  for (p = string, i = 2, sum = 0; *p != EOS; p++, i +=
2)
                        sum += i * *p;
                  return sum % NH_MAX_VAR_HASH_TABLE_NODES;
            } // hash

      private:

};


#endif
```

```
//    File:  NHVariantTable.cpp
//
//    Description:
//
//          Implementation to the NHVariantTable class.
//
//
//    History:
//
//          5/14/97     EFB         Created
//          3/20/98     EFB         Changed names to NH from SN
//


#include <string.h>
#include <stdio.h>

#include    "NHVariantTable.hpp"
#include    "NH_util.hpp"
#include    "NH_culture_codes.h"


NHVariantTable::NHVariantTable(NH_VARIANT_TABLE_TYPES tableType)
{
      status = NH_SUCCESS;
      dirty = false;
            //    clear out the hash table
      for (int i = 0; i < NH_MAX_VAR_HASH_TABLE_NODES; i++)
            variantHashTable[i] = NULL;

      //    initialize our variant id variable.
      nextAvailableVarId = 0;

/*    gnv test stuff
            addVariant("ED", "EDWARD", 0.7, "E ");
            addVariant("GERRY", "GENERIC", 0.7, "G ");
            addVariant("HOP", "HOPSING", 0.7, "C ");
            addVariant("NASSIR", "NARADMAN", 0.7, "A ");
            addVariant("BORRIS", "NATASIA", 0.7, "R ");
            addVariant("JUAN", "EPSTEIN", 0.7, "H ");
            addVariant("KORY", "KOREAN", 0.7, "H ");
*/

/*    snv test stufff
            addVariant("HUANG", "WONG", 0.7, "C ");
*/

      //    the following include lines are commented out because it
takes forever
      //    to compile release versions when they are left in.
      if (tableType == NH_GIVENNAME_VARIANTS)    {
//          #include "gnvdata.h"
      }
      else if (tableType == NH_SURNAME_VARIANTS)       {
//          #include "snvdata.h"
      }
}
```

```
//    release all the memory used to store NH_VarHashTableRecord
pointers
NHVariantTable::~NHVariantTable()
{
      NH_VarHashTableRecordPtr       prevRecord;
      NH_VarHashTableRecordPtr       varRecord;
      unsigned int                                        tableIndex;

      for (tableIndex = 0; tableIndex < NH_MAX_VAR_HASH_TABLE_NODES;
tableIndex++)      {
            varRecord = variantHashTable[tableIndex];
            while (varRecord != NULL)      {
                  prevRecord = varRecord;
                  varRecord = varRecord->next;
                  //    delete the record we allocated,
                  //    as well as the SNVariant object pointed to by
the
                  //    variant member of this record
                  delete prevRecord->variant;
                  delete prevRecord;
            }
      }
}


//    returns the NHVariant object associated with the name,
//    or NULL is there is no object for the name.
NHVariant   *     NHVariantTable::getVariantObjectForName(char *name)
{
      NHVariant                                                *varia
ntObject = NULL;
      unsigned int                                        tableIndex;
      NH_VarHashTableRecordPtr       tempRecordPtr;


      //    find the hash value for the (possible) variant
      tableIndex = hash(name);

      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the variant we are looking for.
      tempRecordPtr = variantHashTable[tableIndex];
      while (tempRecordPtr != NULL) {
            if (!strcmp(tempRecordPtr->segment, name))      {
                  variantObject = tempRecordPtr->variant;
                  break;
            }
            else        //    move on to next record in the chain
                  tempRecordPtr = tempRecordPtr->next;
      }
      return variantObject;
}


//    returns the NHVariant object associated with the name,
//    or creates a new one.
NHVariant   *     NHVariantTable::getOrCreateVariantObjectForName(char
*name)
{
      NHVariant   *variantObject = getVariantObjectForName(name);
```

```
        if (variantObject == NULL)      {
                //      no object existed before, so create one and add it
                //      to the hash table.

                unsigned
int                                                     tableIndex;
                NH_VarHashTableRecordPtr        prevRecord;
                NH_VarHashTableRecordPtr        newVariantHashTableRecord =
new NH_VarHashTableRecord;


                variantObject = new NHVariant(nextAvailableVarId++);
                if (variantObject != NULL)      {
                        //      find the hash value for the name
                        tableIndex = hash(name);

                        //      fill up the values in the record
                        strncpy(newVariantHashTableRecord->segment, name,
NH_MAX_VARIANT_LEN);
                        newVariantHashTableRecord->segment[NH_MAX_VARIANT_LEN]
= EOS;
                        newVariantHashTableRecord->variant = variantObject;
                        newVariantHashTableRecord->next = NULL;

                        //      now add the new record to the chain of entries
                        //      at that index.
                        prevRecord = variantHashTable[tableIndex];
                        if (prevRecord == NULL)
                                variantHashTable[tableIndex] =
newVariantHashTableRecord;
                        else    {
                                while (prevRecord->next != NULL)        {
                                        prevRecord = prevRecord->next;
                                }
                                prevRecord->next = newVariantHashTableRecord;
                        }
                }
                else
                        status = NH_VARIANT_ALLOC_ERROR;
        }
        return variantObject;

}


//      returns the NHVarId associated with the name.  If there is
//      no variant for the name, the function returns NH_VAR_NOT_FOUND.
NHVarId                 NHVariantTable::getVariantIdForName(char *name)
{
        NHVariant               *variantObject = getVariantObjectForName(name);
        NHVarId                         returnId;

        if (variantObject != NULL)      {
                returnId = variantObject->getVariantId();
        }
        else
                returnId = NH_VAR_NOT_FOUND;

        return returnId;
}
```

```
//    Add a variant relationship.
//    In order to do this, we must:
//
//          -     make sure both names already have entries in the hash
table
//                and if not, create them.
//          -     get the id of each entry.
//          -     add the id of each item to the variant information of
the other.
//
//    We handle the special case where the second name is a *.  This
means
//    that the name should be part of the variant table, but not related
//    to anything.  In this case,
//    we only create (or get) a NHVariant object for the name.
NHReturnCode      NHVariantTable::addVariant(char *name1, char *name2,
double varScore,

                                                 char *cultureCode)
{
      NHReturnCode      rc = NH_SUCCESS;
      NHVariant              *varObject1;
      NHVariant              *varObject2;


      if ((varScore < 0.0) || (varScore > 1.0))
            rc = NH_INVALID_VARIANT_SCORE;
      else  {
            if (NH_validate_culture_code(cultureCode))      {

                  //    Get variant object for both names.  This will
also create
                  //    a new entry if the name(s) were not in the table
already
                  varObject1 = getOrCreateVariantObjectForName(name1);

                  //    if the second name was a *, skip the creation of
the second
                  //    NHVariant object and do not associate the names.
                  if (strcmp(name2, "*")) {
                        varObject2 =
getOrCreateVariantObjectForName(name2);

                        if ((varObject1 != NULL) && (varObject2 !=
NULL))      {
                              //    now associate each with the other,
using the supplied variant type
                              rc = varObject1->addVariant(varObject2,
cultureCode, varScore);
                              if (rc == NH_SUCCESS)
                                    rc = varObject2-
>addVariant(varObject1, cultureCode, varScore);
                        }
                  }
            }
            else  {
                  //    flag it as an error, but do not mark the entire
table as bad
                  rc = NH_INVALID_CULTURE_CODE;
            }
      }

      return rc;
}


//    function to change the score associated with two variants.
//    The function return:
//
//          NH_SUCCESS - if things worked out OK
//          NH_VARIANTS_DONT_EXIST - if the either name does not exist
in the table
//
                  or the names are not already variants of each
//
                  other
//          NH_INVALID_VARIANT_SCORE - if the score is invalid
NHReturnCode      NHVariantTable::changeVariantScore(char *name1, char
*name2, char *cultureCode, double newScore)
{
      NHReturnCode      rc = NH_SUCCESS;

      if ((newScore < 0.0) || (newScore > 1.0))
            rc = NH_INVALID_VARIANT_SCORE;
      else  {
            NHVariant   *var1 = getVariantObjectForName(name1);
            NHVariant   *var2 = getVariantObjectForName(name2);

            if ((var1 == NULL) || (var2 == NULL))
                  rc = NH_VARIANTS_DONT_EXIST;
            else  {
                  rc = var1->setVariantScoreForIdAndCulture(var2-
>getVariantId(), cultureCode, newScore);
                  if (rc == NH_SUCCESS)
                        rc = var2->setVariantScoreForIdAndCulture(var1-
>getVariantId(), cultureCode, newScore);
                        //    we should never have a case where the
items are related
                        //    in one direction but not the other.
            }
      }

      return rc;
}


//    a function to remove the relationship between two variants.
//    If either variant ends up without a relationship after this
//    operation, it is left in, but when saved, the resulting file
//    will contain a "*" rather than a related name.  The function can
//    return
//
//          NH_SUCCESS - if things worked out OK
//          NH_VARIANTS_DONT_EXIST - if the names are not already
variants
NHReturnCode      NHVariantTable::removeVariantRelation(char *name1,
char *name2, char *cultureCode)
{
      NHReturnCode      rc = NH_VARIANTS_DONT_EXIST;
      NHVariant               *var1 = getVariantObjectForName(name1);
      NHVariant               *var2 = getVariantObjectForName(name2);

      if ((var1 == NULL) || (var2 == NULL))
            rc = NH_VARIANTS_DONT_EXIST;
      else  {
            if (var1->removeVariant(var2->getVariantId(), cultureCode)
== NH_SUCCESS)    {
                  //    we should never have a case where the items are
related
                  //    in one direction but not the other:
                  if (var2->removeVariant(var1->getVariantId(),
cultureCode) == NH_SUCCESS)
                        rc = NH_SUCCESS;
            }
      }

      return rc;
}
```

```
//	File:  NHVariant.hpp
//
//	Description:
//
//		Interface to the NHVariant class.
//
//
//	History:
//
//		6/6/97		EFB		Created
//		6/23/97		EFB		Changed processing to get rid of
variant types
//								as assign an
individual score for each variant pair.
//		9/9/97		EFB		Changed object so that each
relationship has an
//								associated
culture.  Several access methods have
//								been changed to
allow for a culture specifier.
//		3/20/98		EFB		Changed names to NH from SN
//


/*
	Variant represents the variant information for one name.

	Currently, the name must be a single segment.

	The object contains the following information:

	NHVarId				id;
									//	unique id
for this variant
	byte				numRelatedVariants;
					//	how many variants are we related to?
	NHVarId				variantIds[MAX_VARIANTS_PER_NAME];//
what are the id's of our related variants
	double				varScores[MAX_VARIANTS_PER_NAME];	//
Score for each variant


//	in variants array above
	short int		varCultures[MAX_VARIANTS_PER_NAME]; //	Two
byte code describing the culture


	//	for this variant relationship.  These are


	//	actually char[2] codes.

	A variant knows how to add an id, type combination to its
information.
*/

#ifndef	NHVARIANT_HPP
#define NHVARIANT_HPP

#include	<stdlib.h>
```

```
#include     "NH_get_error_text.h"
#include     "NH_culture_codes.h"


typedef      unsigned char byte;


//    #define            MAX_VARIANTS_PER_NAME          30
#define            NH_INIT_VARIANTS_PER_NAME          5



//    define a constant to represent that two variants were
//    not related.
#define      NH_VARIANTS_NOT_RELATED        -1.0

//    define a variant id as a short int.
typedef      short int   NHVarId;

#define            NH_VAR_NOT_FOUND                           -1


//    define      a structure to hold the info about a related variant.
We
//    will use arrays of this structure to list the names related to
//    a variant.
typedef struct NH_RELATED_VARIANTS_T       {
      NHVarId                 variantId;  //    what is the id of our
related variant
      double                  varScore;         //    Score for this
variant, as related to the main variant


      //    in variants array above
      char                    varCulture[NH_MAX_CULTURE_CODE_LEN];
//    Two byte code describing the culture


            //    for this variant relationship.  These are


            //    actually char[2] codes.
} NH_RelatedVariants;



class NHVariant
{
      public:
            NHVariant(NHVarId newId);
            virtual     ~NHVariant();



            //    Returns the variant score for the relationship between
the
            //    the supplied variant id and the variant, within the
specified
            //    culture.  If the variants are not related, the
function returns
```

```
            //    NH_VARIANTS_NOT_RELATED.
            double getVariantScoreForIdAndCulture(NHVarId relatedVarId,
char *cultCode);

            //    allows caller to search for across cultures within
this variant
            double     getVariantScoreForIdAndAnyCulture(NHVarId
relatedVarId, char *cultCode);
            //    see if the supplied variant is related to us, and if
so,
            //    replace the existing score with the new score.
            //    if not, return NH_VARIANTS_DONT_EXIST.
            NHReturnCode setVariantScoreForIdAndCulture(NHVarId
relatedVarId,


char *cultCode, double score);

            //    adds the id of the specified variant (along with an
associated
            //    score and culture code)  to our array of variants
related to us.
            virtual    NHReturnCode      addVariant(NHVariant *variant,
char *cultureCode,

                                          double relatedVarScore);

            //    remove a variant from our list
            //    return NH_VARIANTS_DONT_EXIST if the id is not in our
list already
            virtual NHReturnCode    removeVariant(NHVarId relatedVarId,
char *cultureCode);

            //    return the variant id for this object
            NHVarId     getVariantId()    {return id;}

            //    return the variant id for this object
            byte  getNumVariants()  {return numRelatedVariants;}

            NHVarId     getIdForRelatedVariant(int relVarIndex)
            {
                  NHVarId     varId = 0;


                  if ((relVarIndex > -1) && (relVarIndex <
numRelatedVariants))
                        varId = relatedVariants[relVarIndex].variantId;
                  return varId;
            }


            char  *     getCultureCodeForRelatedVariant(int relVarIndex)
            {
                  char  *cultureCode = NULL;


                  if ((relVarIndex > -1) && (relVarIndex <
numRelatedVariants))
                        cultureCode =
relatedVariants[relVarIndex].varCulture;
                  return cultureCode;
            }

            double      getScoreForRelatedVariant(int relVarIndex)
            {
                  double      score = 0.0;

                  if ((relVarIndex > -1) && (relVarIndex <
numRelatedVariants))
                        score = relatedVariants[relVarIndex].varScore;
                  return score;
            }

      protected:

            NHVarId                                                 id;
                                                              //    unique id
for this variant
            byte                                                    numRelatedVa
riants;                 //    how many variants are we related to?
            byte                                                    maxRelatedVa
riants;                 //    how many variants are we related to?
            NH_RelatedVariants      *relatedVariants;

      private:

};


#endif
```

```
//     File:  NHVariant.cpp
//
//     Description:
//
//           Implementation to the NHVariant class.
//
//
//     History:
//
//           6/6/97      EFB         Created
//           3/20/98     EFB         Changed names to NH from SN
//


#include <string.h>
#include <stdio.h>


#include    "NHVariant.hpp"
#include    "NH_util.hpp"


#ifndef     false
#define     false 0
#endif

#ifndef     true
#define     true  1
#endif



NHVariant::NHVariant(NHVarId newId)
{
      id = newId;
      numRelatedVariants = 0;
      maxRelatedVariants = NH_INIT_VARIANTS_PER_NAME;
      relatedVariants = new NH_RelatedVariants[maxRelatedVariants];
}


NHVariant::~NHVariant()
{
      if (relatedVariants)
            delete [] relatedVariants;
}


//    see if the supplied variant is related to us, and if so, return
its score.
double      NHVariant::getVariantScoreForIdAndCulture(NHVarId
relatedVarId, char *cultCode)
{
double      returnScore = NH_VARIANTS_NOT_RELATED;

      for (int i = 0; i < numRelatedVariants; i++)     {
            if ((relatedVariants[i].variantId == relatedVarId) &&
(memcmp(relatedVariants[i].varCulture, cultCode,
NH_MAX_CULTURE_CODE_LEN) == 0))      {
                  returnScore = relatedVariants[i].varScore;
                  break;
            }
      }
      return returnScore;
}


//    See if the supplied variant is related to us under any culture.
//    Because this method is intended to be called several times (for
//    possibly multiple cultures, it also takes a culture string that
//    is used to keep track of the last culture that was returned.  The
//    first time the function is called, the culture is specified as an
//    empty string.  On return, it contains the first culture found
//    in the list for the id.  The next time the function is called,
//    we look past that culture/id combination in the array looking for
//    the next one, until we return NH_VARIANTS_NOT_RELATED.
double      NHVariant::getVariantScoreForIdAndAnyCulture(NHVarId
relatedVarId, char *cultCode)
{
      double      returnScore = NH_VARIANTS_NOT_RELATED;
      bool        alreadyFoundLastCultCode = false;

      for (int i = 0; i < numRelatedVariants; i++)      {
            if ((relatedVariants[i].variantId == relatedVarId))     {
                  //    ids matched, so see if they specified a culture
code
                  if (*cultCode == EOS)     {
                        //    this is first time through, so no check is
necessary.
                        //    copy the cult code into the supplied
string.
                        NH_safe_strcpy(cultCode,
relatedVariants[i].varCulture, NH_MAX_CULTURE_CODE_LEN);
                        returnScore = relatedVariants[i].varScore;
                        break;
                  }
                  else  {
                        //    this is not first time through, they are
passing us the cult code
                        //    that was found last time, so see if we
have already found that one
                        if (alreadyFoundLastCultCode == true)      {
                              NH_safe_strcpy(cultCode,
relatedVariants[i].varCulture, NH_MAX_CULTURE_CODE_LEN);
                              returnScore = relatedVariants[i].varScore;
                              break;
                        }
                        else  {
                              //    see if this is the cult code they
passed us
                              if (memcmp(relatedVariants[i].varCulture,
cultCode, NH_MAX_CULTURE_CODE_LEN) == 0)  {
                                    alreadyFoundLastCultCode =
true;       //    we found it
                              }
                        }
                  }
            }
      }
      return returnScore;
}
```

```
//    see if the supplied variant is related to us, and if so,
//    replace the existing score with the new score.
//    if not, return NH_VARIANTS_DONT_EXIST.
NHReturnCode       NHVariant::setVariantScoreForIdAndCulture(NHVarId
relatedVarId,

                               char *cultCode, double score)
{
NHReturnCode       rc = NH_VARIANTS_DONT_EXIST;

        for (int i = 0; i < numRelatedVariants; i++)     {
                if ((relatedVariants[i].variantId == relatedVarId) &&
                                (memcmp(relatedVariants[i].varCulture, cultCode,
NH_MAX_CULTURE_CODE_LEN) == 0))      {
                        relatedVariants[i].varScore = score;
                        rc = NH_SUCCESS;
                        break;
                }
        }
        return rc;
}

//    add a variant to our list
//    if the variant is already in the list, do not add it a second
//    time, and return an error
NHReturnCode       NHVariant::addVariant(NHVariant *variant, char
*cultureCode,

                                          double relatedVarScore)
{
        NHReturnCode       rc = NH_SUCCESS;
        NHVarId relatedVarId = variant->getVariantId();


        //    check to see if the relationship has already been
        //    defined for this id/culture.
        for (int i = 0; i < numRelatedVariants; i++)     {
                if ((relatedVariants[i].variantId == relatedVarId) &&
                                (memcmp(relatedVariants[i].varCulture,
cultureCode, NH_MAX_CULTURE_CODE_LEN) == 0))     {
                        rc = NH_VARIANT_ALREADY_RELATED;
                        break;
                }
        }

        if (rc == NH_SUCCESS)    {
                //    see if we are maxed out
                if (numRelatedVariants == maxRelatedVariants)    {
                        //    try to reallocate the space
                        NH_RelatedVariants       *biggerBlock;

                        biggerBlock = new
NH_RelatedVariants[maxRelatedVariants * 2];

                        if (biggerBlock)  {
                                memcpy(biggerBlock, relatedVariants,
                                                sizeof(NH_RelatedVariant
s) * maxRelatedVariants);
                        delete [] relatedVariants;
                        relatedVariants = biggerBlock;
                        maxRelatedVariants *= 2;
                  }
                  else
                        rc = NH_MAX_VARIANT_SIZE_INCREMENT_FAILED;
            }
      }

      if (rc == NH_SUCCESS)   {
            relatedVariants[numRelatedVariants].variantId =
relatedVarId;
            relatedVariants[numRelatedVariants].varScore =
relatedVarScore;
            strncpy(relatedVariants[numRelatedVariants].varCulture,
cultureCode, NH_MAX_CULTURE_CODE_LEN);
            numRelatedVariants++;
      }

      return rc;
}


//    remove a variant from our list
//    return NH_VARIANTS_DONT_EXIST if the id is not in our list already
NHReturnCode      NHVariant::removeVariant(NHVarId relatedVarId, char
*cultureCode)
{
NHReturnCode      rc = NH_VARIANTS_DONT_EXIST;

      for (int i = 0; i < numRelatedVariants; i++)    {
            if ((relatedVariants[i].variantId == relatedVarId) &&
                        (memcmp(relatedVariants[i].varCulture,
cultureCode, NH_MAX_CULTURE_CODE_LEN) == 0))    {
                  //    now move any ids past the one that match
                  //    back one space.
                  for (int j = i + 1; j < numRelatedVariants;
j++)  {
                        relatedVariants[j - 1].varScore =
relatedVariants[j].varScore;
                        relatedVariants[j - 1].variantId =
relatedVariants[j].variantId;
                        strncpy(relatedVariants[j - 1].varCulture,
                                                relatedVariants[j].varCu
lture, NH_MAX_CULTURE_CODE_LEN);
                  }
                  numRelatedVariants--;         //    we not have one
less variant
                  rc = NH_SUCCESS;
                  break;
            }
      }
      return rc;
}
```

```
//     File:  NHTAQTable.hpp
//
//     Description:
//
//           Interface to the NHTAQTable class.
//
//
//     History:
//
//           5/7/97      EFB          Created
//           3/20/98     EFB          Changed names to NH from SN
//
//
//
//     The TAQTable is organized by name and culture.  That is the unique
key
//     in the table.  We do lookups by hashing the name, but must
consider the
//     culture code as we walk the hash table bucket.

#ifndef     NHTAQTABLE_HPP
#define NHTAQTABLE_HPP


#include    "NH_culture_codes.h"
#include    "NHNameData.hpp"
#include    "NH_get_error_text.h"



//    how many characters can a TAQ value be?
#define            NH_MAX_TAQ_LEN          20


//    define the possible values for the TAQ action
#define            NH_TAQ_ACTION_DELETE                 'X'
#define            NH_TAQ_ACTION_DISREGARD       'D'


//    define a record in the hash table of TAQ values
typedef struct NH_TAQ_RECORD_T      {
      char    taqString[NH_MAX_TAQ_LEN + 1];     //    string that is the
TAQ value
      char        taqType;               //    P, S, I, T or Q
      char        sepIfConjoined;        //    Y or N
      char        gnAction;                    //    what to do when
found in gn
      char        snAction;                    //    what to do when
found in sn
      char        taqCulture[NH_MAX_CULTURE_CODE_LEN +
1];                 //    which culture does this apply to?
  struct NH_TAQ_RECORD_T *next;          //    pointer to next TAQ
record in this hash branch
} NH_TAQRecord;

//    Do not change without seeing function NH_TAQhash().
#define NH_MAX_TAQ_HASH_NODES 907

//    define a type that is a pointer to a NH_TAQRecord
typedef NH_TAQRecord *NH_TAQRecordPtr;
```

```
//    define a type that is a table (array) of NH_TAQRecordPtrs
typedef NH_TAQRecordPtr NH_TAQHashTable[NH_MAX_TAQ_HASH_NODES];


enum  NH_TAQ_TABLE_TYPE {
      NH_PRODUCTION_TAQ_TABLE,
      NH_EMPTY_TAQ_TABLE
};


class NHTAQTable
{
      public:
            NHTAQTable(NH_TAQ_TABLE_TYPE type);
            ~NHTAQTable();


            //    function to return a pointer to the TAQ structure for
the
            //    supplied character string (segment), cultureCode
combination.
            //    Returns NULL if the supplied segment is not known to
the TAQ table
            //    for the specified culture code.
            NH_TAQRecordPtr               getTAQSegment(char *nameSeg,
char *cultureCode);

            //    specialized version of the above function that looks
for the
            //    name segment in either of the specified culture codes.
It makes
            //    sure that if the name is found in the
primaryCultureCode, that one
            //    gets returned even if we come upon the
secondaryCultureCode first.
            NH_TAQRecordPtr               getTAQSegment(char *nameSeg,
char *primaryCultureCode,

                                          char *secondaryCultureCode);

            NHReturnCode                        getStatus() {return
status;}

            bool                                            getDirty()
{return dirty;}
            void                                            setDirty(boo
l aBool)    {dirty = aBool;}

            int                                                   getNum
HashBuckets()     {return NH_MAX_TAQ_HASH_NODES;}

            NH_TAQRecordPtr               getHashBucketStartNodeAt(int
hashTableIndex)

                        {return taqHashTable[hashTableIndex];}

            NHReturnCode                        addTAQValue(char
*taqValue, char taqType,

                                    char sepIfConjoined, char
gnTAQAction,
                                        char snTAQAction, char *taqCulture);

            NHReturnCode                      removeTAQValue(char
*taqValue, char *cultureCode);

      protected:

      private:

            //    Returns an integer in the range [0,
NH_MAX_TAQ_HASH_NODES].
            inline unsigned int     hash(char *string)
            {
                  char                          *p;
                  unsigned int      i;
                  unsigned int      sum;

                  for (p = string, i = 2, sum = 0; *p != EOS; p++, i +=
2)
                        sum += i * *p;
                  return sum % NH_MAX_TAQ_HASH_NODES;
            } /* hash */

        NH_TAQHashTable taqHashTable;
            NHReturnCode            status;                 //    are we
valid
            bool              dirty;            //    have we changed



};


#endif
```

```
//    File:  NHTAQTable.cpp
//
//    Description:
//
//          Implementation to the NHTAQTable class.
//
//
//    History:
//
//          5/14/97     EFB         Created
//          9/9/97      EFB         Added support for culture
//          3/20/98     EFB         Changed names to NH from SN
//


#include <string.h>
#include <stdio.h>


#include    "NHTAQTable.hpp"
#include    "NH_util.hpp"


NHTAQTable::NHTAQTable(NH_TAQ_TABLE_TYPE type)
{
      status = NH_SUCCESS;

      //    clear out the hash table
      for (int i = 0; i < NH_MAX_TAQ_HASH_NODES; i++)
            taqHashTable[i] = NULL;

      //    make sure we are not supposed to be doing an empty table.
      if (type == NH_PRODUCTION_TAQ_TABLE)      {
            //    parameters are:
            //
            //                TAQ string
            //                taq Type (T, P, S, Q, I),
            //                sepIfConjoined    ('Y' or 'N')
            //                Given name action       (D - delete, R -
disregard, X - not applicable)
            //                Surname action          (D - delete, R -
disregard, X - not applicable)
            //                Culture  (2 char code)

            //    include the data that was generated via the TAQmanager
tool.
            #include    "taqdata.h"

            //    This stuff is just left over from testing
      /*
            addTAQValue("DR", 'T', 'N', NH_TAQ_ACTION_DELETE,
NH_TAQ_ACTION_DELETE, NH_CULTURE_CODE_GENERIC);
            addTAQValue("MR", 'T', 'N', NH_TAQ_ACTION_DELETE,
NH_TAQ_ACTION_DELETE, NH_CULTURE_CODE_GENERIC);
            addTAQValue("MRS", 'T', 'N', NH_TAQ_ACTION_DELETE,
NH_TAQ_ACTION_DELETE, NH_CULTURE_CODE_GENERIC);
            addTAQValue("JR", 'Q', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_GENERIC);
            addTAQValue("SR", 'Q', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_GENERIC);
      addTAQValue("ABDUL", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_ARABIC);
      addTAQValue("HOMEY", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_ANGLO);
      addTAQValue("CHINTAQ", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_CHINESE);
      addTAQValue("HISPTAQ", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_HISPANIC);
      addTAQValue("KORTAQ", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_KOREAN);
      addTAQValue("RUSTAQ", 'T', 'N', NH_TAQ_ACTION_DISREGARD,
NH_TAQ_ACTION_DISREGARD, NH_CULTURE_CODE_RUSSIAN);

   */
   }

   //    mark that the table has not been changed.   Usefull for
TAQManager application
   dirty = false;
}



//    release all the memory used to store the NH_TAQRecords
NHTAQTable::~NHTAQTable()
{
   NH_TAQRecord      *prevTAQRecord;
   NH_TAQRecord      *taqRecord;
   int                         tableIndex;

   for (tableIndex = 0; tableIndex < NH_MAX_TAQ_HASH_NODES;
tableIndex++)      {
      taqRecord = taqHashTable[tableIndex];
      while (taqRecord != NULL)      {
         prevTAQRecord = taqRecord;
         taqRecord = taqRecord->next;
         delete prevTAQRecord;
      }
   }
}


//    function to take the values passed in, create a NH_TAQRecord
//    structure, and add the new structure to this object's
//    taqHashTable.
NHReturnCode NHTAQTable::addTAQValue(char *taqValue, char taqType, char
sepIfConjoined,

         char gnTAQAction, char snTAQAction, char *taqCulture)
{
   NHReturnCode      rc = NH_SUCCESS;
   NH_TAQRecord      *newTAQRecord;
   int                         tableIndex;
   NH_TAQRecord      *prevTAQRecord;


   //    first, make sure we know the culture code
   if (NH_validate_culture_code(taqCulture)) {
      //    find the hash value for the taq
      tableIndex = hash(taqValue);
```

```
            //    now see if the taq is already defined for this culture
code
            //    At the same time, find our insertion point, which will
be either:
            //          -     the first node in the bucket, if this
bucket is empty
            //          -     the end of the bucket
            prevTAQRecord = taqHashTable[tableIndex];
            if (prevTAQRecord != NULL)     {
                  rc = NH_TAQ_ALREADY_EXISTS;                //    assume
it exists
                  while (strcmp(prevTAQRecord->taqString, taqValue) ||
                                          (strcmp(prevTAQRecord-
>taqCulture, taqCulture)))     {
                        if (prevTAQRecord->next == NULL)          {
                              rc = NH_SUCCESS;               //    does
not exist, so looks good so far
                              break;           //    end of bucket
chain
                        }
                        prevTAQRecord = prevTAQRecord-
>next;            //    move though bucket chain
                  }
            }

            //    if all is still ok (e.g. no duplicate)
            if (rc == NH_SUCCESS)    {
                  //    now create the new record and set its values
                  newTAQRecord = new NH_TAQRecord;
                  if (newTAQRecord != NULL)     {
                        NH_safe_strcpy(newTAQRecord->taqString,
taqValue, NH_MAX_TAQ_LEN);
                        newTAQRecord->taqType = taqType;
                        newTAQRecord->sepIfConjoined = sepIfConjoined;
                        newTAQRecord->gnAction = gnTAQAction;
                        newTAQRecord->snAction = snTAQAction;
                        NH_safe_strcpy(newTAQRecord->taqCulture,
taqCulture, NH_MAX_CULTURE_CODE_LEN);
                        newTAQRecord->next = NULL;

                        //    now add the new record to the chain of
entries (or the start of the
                        //    bucket.  We have already hashed the
tableIndex value above, and have
                        //    found the correct insertion point
                        if (prevTAQRecord == NULL)
                              taqHashTable[tableIndex] = newTAQRecord;
                        else
                              prevTAQRecord->next = newTAQRecord;
                  }
                  else  {
                        rc = NH_TAQ_RECORD_ALLOC_ERROR;
                        status = NH_TAQ_RECORD_ALLOC_ERROR;
                  }
            }
      }
      else  {
            //    flag it as an error, but do not mark the entire table
as bad
            rc = NH_INVALID_CULTURE_CODE;
      }

      return rc;
}


NH_TAQRecordPtr NHTAQTable::getTAQSegment(char *nameSeg, char
*cultureCode)
{
      int                                       tableIndex;
      NH_TAQRecordPtr   tempTAQRecordPtr;
      NH_TAQRecordPtr   returnTAQRecordPtr = NULL;


      //    find the hash value for the (possible) taq
      tableIndex = hash(nameSeg);

      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the taq we are looking for.
      tempTAQRecordPtr = taqHashTable[tableIndex];
      while (tempTAQRecordPtr != NULL)     {
            if (!strcmp(tempTAQRecordPtr->taqString, nameSeg) &&
                              !strcmp(tempTAQRecordPtr->taqCulture,
cultureCode))      {
                  returnTAQRecordPtr = tempTAQRecordPtr;
                  break;
            }
            else        //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
      }
      return returnTAQRecordPtr;
}


//    specialized version of the above function that looks for the
//    name segment in either of the specified culture codes.  It makes
//    sure that if the name is found in the primaryCultureCode, that one
//    gets returned even if we come upon the secondaryCultureCode first.
NH_TAQRecordPtr               NHTAQTable::getTAQSegment(char *nameSeg,
char *primaryCultureCode,

                              char *secondaryCultureCode)
{
      int                                       tableIndex;
      NH_TAQRecordPtr   tempTAQRecordPtr;
      NH_TAQRecordPtr   returnTAQRecordPtr = NULL;


      //    find the hash value for the (possible) taq
      tableIndex = hash(nameSeg);

      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the taq we are looking for.
      tempTAQRecordPtr = taqHashTable[tableIndex];
      while (tempTAQRecordPtr != NULL)     {
            if (!strcmp(tempTAQRecordPtr->taqString, nameSeg) &&
                              !strcmp(tempTAQRecordPtr->taqCulture,
primaryCultureCode))     {
                  returnTAQRecordPtr = tempTAQRecordPtr;
                  break;
            }
            else        //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
      }

      //    see if we need to check the secondary
      if (returnTAQRecordPtr == NULL)    {
            //    go throught the records in the chain at that offset in
the
            //    hash table, and try to find the taq we are looking
for.
            tempTAQRecordPtr = taqHashTable[tableIndex];
            while (tempTAQRecordPtr != NULL)    {
                  if (!strcmp(tempTAQRecordPtr->taqString, nameSeg) &&
                                    !strcmp(tempTAQRecordPtr-
>taqCulture, secondaryCultureCode)) {
                        returnTAQRecordPtr = tempTAQRecordPtr;
                        break;
                  }
                  else        //    move on to next record in the chain
                        tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      return returnTAQRecordPtr;
}

//    try to remove the TAQ value specified.  If found, return
//    NH_SUCCESS. If not found, return.
//    The record is deleted if found.
NHReturnCode            NHTAQTable::removeTAQValue(char *taqValue, char
*cultureCode)
{
      NHReturnCode            rc = NH_TAQ_NOT_FOUND;
      NH_TAQRecordPtr   tempTAQRecordPtr;
      NH_TAQRecordPtr   prevTAQRecordPtr = NULL;
      int                                     tableIndex =
hash(taqValue);


      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the taq we are looking for.
      tempTAQRecordPtr = taqHashTable[tableIndex];
      while (tempTAQRecordPtr != NULL)    {
            if (!strcmp(tempTAQRecordPtr->taqString, taqValue) &&
                              !strcmp(tempTAQRecordPtr->taqCulture,
cultureCode))
                  break;
            else  {
                  //    save this as the prev
                  prevTAQRecordPtr = tempTAQRecordPtr;
                  //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      //    once we are here, tempTAQRecordPtr will be non NULL
      //    if we found it.
      if (tempTAQRecordPtr != NULL) {
            if (prevTAQRecordPtr == NULL) {
                  //    this record was the first in the chain, so we
must alter
```

```
            else        //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
      }

      //    see if we need to check the secondary
      if (returnTAQRecordPtr == NULL)     {
            //    go throught the records in the chain at that offset in
the
            //    hash table, and try to find the taq we are looking
for.
            tempTAQRecordPtr = taqHashTable[tableIndex];
            while (tempTAQRecordPtr != NULL)     {
                  if (!strcmp(tempTAQRecordPtr->taqString, nameSeg) &&
                                    !strcmp(tempTAQRecordPtr-
>taqCulture, secondaryCultureCode)) {
                        returnTAQRecordPtr = tempTAQRecordPtr;
                        break;
                  }
                  else        //    move on to next record in the chain
                        tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      return returnTAQRecordPtr;
}

//    try to remove the TAQ value specified.  If found, return
//    NH_SUCCESS. If not found, return.
//    The record is deleted if found.
NHReturnCode            NHTAQTable::removeTAQValue(char *taqValue, char
*cultureCode)
{
      NHReturnCode            rc = NH_TAQ_NOT_FOUND;
      NH_TAQRecordPtr   tempTAQRecordPtr;
      NH_TAQRecordPtr   prevTAQRecordPtr = NULL;
      int                                       tableIndex =
hash(taqValue);


      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the taq we are looking for.
      tempTAQRecordPtr = taqHashTable[tableIndex];
      while (tempTAQRecordPtr != NULL)     {
            if (!strcmp(tempTAQRecordPtr->taqString, taqValue) &&
                              !strcmp(tempTAQRecordPtr->taqCulture,
cultureCode))
                  break;
            else  {
                  //    save this as the prev
                  prevTAQRecordPtr = tempTAQRecordPtr;
                  //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      //    once we are here, tempTAQRecordPtr will be non NULL
      //    if we found it.
      if (tempTAQRecordPtr != NULL) {
            if (prevTAQRecordPtr == NULL) {
                  //    this record was the first in the chain, so we
must alter
```

```
            else        //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
      }

      //    see if we need to check the secondary
      if (returnTAQRecordPtr == NULL)     {
            //    go throught the records in the chain at that offset in
the
            //    hash table, and try to find the taq we are looking
for.
            tempTAQRecordPtr = taqHashTable[tableIndex];
            while (tempTAQRecordPtr != NULL)     {
                  if (!strcmp(tempTAQRecordPtr->taqString, nameSeg) &&
                                    !strcmp(tempTAQRecordPtr-
>taqCulture, secondaryCultureCode)) {
                        returnTAQRecordPtr = tempTAQRecordPtr;
                        break;
                  }
                  else        //    move on to next record in the chain
                        tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      return returnTAQRecordPtr;
}

//    try to remove the TAQ value specified.  If found, return
//    NH_SUCCESS. If not found, return.
//    The record is deleted if found.
NHReturnCode            NHTAQTable::removeTAQValue(char *taqValue, char
*cultureCode)
{
      NHReturnCode            rc = NH_TAQ_NOT_FOUND;
      NH_TAQRecordPtr   tempTAQRecordPtr;
      NH_TAQRecordPtr   prevTAQRecordPtr = NULL;
      int                                     tableIndex =
hash(taqValue);


      //    go throught the records in the chain at that offset in the
      //    hash table, and try to find the taq we are looking for.
      tempTAQRecordPtr = taqHashTable[tableIndex];
      while (tempTAQRecordPtr != NULL)     {
            if (!strcmp(tempTAQRecordPtr->taqString, taqValue) &&
                              !strcmp(tempTAQRecordPtr->taqCulture,
cultureCode))
                  break;
            else  {
                  //    save this as the prev
                  prevTAQRecordPtr = tempTAQRecordPtr;
                  //    move on to next record in the chain
                  tempTAQRecordPtr = tempTAQRecordPtr->next;
            }
      }

      //    once we are here, tempTAQRecordPtr will be non NULL
      //    if we found it.
      if (tempTAQRecordPtr != NULL) {
            if (prevTAQRecordPtr == NULL) {
                  //    this record was the first in the chain, so we
must alter
```

```
                  //     the hash table entry
                  taqHashTable[tableIndex] = tempTAQRecordPtr->next;
            }
            else        //     not the first in the chain, so assign the
previous one's next
                  prevTAQRecordPtr->next = tempTAQRecordPtr-
>next;      //     to our next
            delete tempTAQRecordPtr;
            rc = NH_SUCCESS;
      }

      return rc;
}
```

```
have
      if ((numGnSegments < NH_MAX_SEGS_BEFORE_TAQ) &&
                  (*(gnSegments[numGnSegments].segString) !=
EOS)) {
            gnSegments[numGnSegments].status =
NH_NAME_FIELD_STATUS_KNOWN;
            *outChar = EOS;         //    terminate the last segment
            numGnSegments++;  //    look at next segment
      }

      //    now do the surname
      numSnSegments = 0;
      inChar = sn;
      outChar = snSegString;
      *outChar = EOS;
      snSegments[0].segString = outChar;
      while ((*inChar != EOS) && (numSnSegments <
NH_MAX_SEGS_BEFORE_TAQ))      {
            //    if this is a noise character, just move on to the next
one in the name
            if (strchr(noiseChars, *inChar))
                  inChar++;
            else  {
                  if (strchr(segDelimChars, *inChar)) {
                        //    make sure this is not the next in a series
of white spaces
                        if (*(snSegments[numSnSegments].segString) !=
EOS)  {
                              snSegments[numSnSegments].status =
NH_NAME_FIELD_STATUS_KNOWN;
                              *outChar = EOS;         //    terminate
the last segment
                              numSnSegments++;  //    look at next
segment
                              //    make sure we are not past the max
number of segments
                              if (numSnSegments >=
NH_MAX_SEGS_BEFORE_TAQ)
                                    break;
                              inChar++;                           //
look at next char in name
                              outChar++;                    //    point
to next available space in the output array
                              snSegments[numSnSegments].segString =
outChar;
                              *outChar = EOS;         // init the new
segment
                        }
                        else        //    this is a segDelim char, and
so was the last one.
                              inChar++;               //    so just
ignore it, and move on
                  }
                  else  {
                        //    just a regular character, so add it to the
segment we are
                        //    working on currently
                        *outChar = toupper(*inChar);
                        outChar++;                    //    write to
next character in segment next time.
                        inChar++;                           //    look
```

```
at next char in name
                  }
            }
      }
      //    if we get here, it is because we reached the end of the sn
string.
      //    If we were in the middle of building a name segment, we
should
      //    terminate the segment and increase the number of segments we
have
      if ((numSnSegments < NH_MAX_SEGS_BEFORE_TAQ) &&
                  (*(snSegments[numSnSegments].segString) !=
EOS)) {
            snSegments[numSnSegments].status =
NH_NAME_FIELD_STATUS_KNOWN;
            *outChar = EOS;         //    terminate the last segment
            numSnSegments++;  //    look at next segment
      }

      //    now see if there are any segments at all
      //    in the fields.  If not, we should create a
      //    single blank segment, and mark its status as
      //    unknown.  If there are segments, we need to check for the
      //    special values NFN, NLN, NMN, FNU, LNU, MNU.  If we find
these,
      //    blank out the segment, and set the status
      //    appropriately.
      //    When a name field has more than one segment, but still
      //    specifies one of these values, we still blank it out,
      //    but we keep the segment as a blank segment.  Although the
      //    digraph score for this segment will be largely determined by
      //    the UNKNOWN or NONE parameter, it still gets treated as a
      //    segment in that oops and anchor val can be applied, and
      //    it still gets sent to best score.
      //    We do not currently look across name fields for these
markers.
      //    That is, we look for NFN, NMN, FNU. MNU in the given name
field
      //    and we look for NLN and LNU in the surname field.
      //    ??? Future versions may look across name fields.

      if (numGnSegments == 0) {
            numGnSegments = 1;
            gnSegments[0].segString = "";
            gnSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
      }
      else if (nameParms->getCheckGnUnknowns()) {
            for (i = 0; i < numGnSegments; i++)       {
                  if (!strcmp(gnSegments[i].segString, "NFN"))      {
                        gnSegments[i].segString[0] = EOS;
                        gnSegments[i].status =
NH_NAME_FIELD_STATUS_NON_EXISTANT;
                  }     else if     (!strcmp(gnSegments[i].segString,
"FNU")){
                        gnSegments[i].segString[0] = EOS;
                        gnSegments[i].status =
NH_NAME_FIELD_STATUS_UNKNOWN;
                  }     else if     (!strcmp(gnSegments[i].segString,
"NMN")){
                        gnSegments[i].segString[0] = EOS;
                        gnSegments[i].status =
NH_NAME_FIELD_STATUS_NON_EXISTANT;
                  }     else if     (!strcmp(gnSegments[i].segString,
"MNU")){
                        gnSegments[i].segString[0] = EOS;
                        gnSegments[i].status =
NH_NAME_FIELD_STATUS_UNKNOWN;
                  }
            }
      }

      //    now the sn segs
      if (numSnSegments == 0) {
            numSnSegments = 1;
            snSegments[0].segString = "";
            snSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
      }
      else if (nameParms->getCheckSnUnknowns()) {
            for (i = 0; i < numSnSegments; i++)       {
                  if    (!strcmp(snSegments[i].segString, "NLN")){
                        snSegments[i].segString[0] = EOS;
                        snSegments[i].status =
NH_NAME_FIELD_STATUS_NON_EXISTANT;
                  }     else if     (!strcmp(snSegments[i].segString,
"LNU"))     {
                        snSegments[i].segString[0] = EOS;
                        snSegments[i].status =
NH_NAME_FIELD_STATUS_UNKNOWN;
                  }
            }
      }
}


//    function to go through the segments and for each one, see if
//    it is a TAQ value.  If so, we associate the TAQ with the previous
//    or following segment, depending on its type (i.e. prefix, suffix,
etc).
//    When we store the TAQ, we also store the action associated with
//    the TAQ (currently DELETE or DISREGARD), since this information
//    will be required to determine how to adjust the base segment score
//
//    Deciding which segment to associate a TAQ with can get pretty
//    hairy, especially when mulitple TAQs can be in a name field
//    consecutively.  We use the Following rules for single TAQ values:
//
//    TAQ Type                Segment to Associate with
//
//    Prefix                        next segment
//    Suffix                        previous segment
//    Infix                         Not supported yet
//    Title                         next segment
//    Qualifier               previous segment
//
//    These are the basic rules for figuring out which segment to
associate
//    TAQs with:
//
//    -     Any TAQ segments before the first Name segment are
associated with
//          the first name segment
```

```
//
//    -     Any TAQ segments after the last Name segment are associated
with
//          the last Name segment
//
//    -     For TAQs that are surrounded by Name segments :
//
//          -     All TAQs between a Name segment (on the left) and a
suffix (qualifier)
//                (on the right) are associated with the Name Segment.
//
//          -     All TAQs not fitting the above are assoicated with the
Name segment
//                they proceed.
//
void  NHNameData::processTAQValues(NHTAQTable *taqTable)
{
      //    NHTAQAction             taqAction;
      int                                     i;
      NH_TAQRecordPtr   tempTAQList[NH_MAX_TAQS_PER_SEGMENT];
//    temp list of TAQs found
      int                                     tempTAQSegIndex;  //
temp index for the tempTaqList
      NH_TAQRecordPtr   tempTAQRecordPtr; //    pointer to structure for
a TAQ record
      int                                     numTempTAQSegs;
//    how many TAQs did we find
      int                                     segIndex;
            //    which segment are we looking at
      int                                     lastPrefixIndex;  //
index of last prefix like segment we got
      int                                     lastSuffixIndex;  //
index of last suffix like segment we got
      int                                     lastNameIndex;
//    index of last non-TAQ segment we got
      int                                     nameSegmentTaqListIndex;
            //    where to put taqs in a name segments taq list
      char                              *primaryCultureCode =
nameParms->primaryCultureCode;
      char                              *secondaryCultureCode =
nameParms->secondaryCultureCode;

      //    clear out the TAQ counts for each segment.
      //    This is important because the TAQ segments are not
initalized
      //    if they are not filled in.
      for (i = 0; i < numGnSegments; i++)
            gnSegments[i].numTAQs = 0;

      if    (nameParms->getSeparateGnTaqs() == true)  {
            //    init some variables
            segIndex = 0;
            numTempTAQSegs = 0;

            //    Start out by looking for TAQs at the start of the name
field,
            //    before any name segments.
            //    while there are TAQ values at the start of the gn
            //    get their associated TAQ record and place that in
            //    a temporary list.
            while (segIndex < numGnSegments)    {
                  tempTAQRecordPtr = taqTable-
>getTAQSegment(gnSegments[segIndex].segString,


      primaryCultureCode,


      secondaryCultureCode);
                  if (tempTAQRecordPtr != NULL) {
                        //    make sure we are not past our space for
TAQs in the temp list
                        //    This would happen if a name field started
out with tons of TAQs
                        if (segIndex < NH_MAX_TAQS_PER_SEGMENT)    {
                              tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                              numTempTAQSegs++;
                        }
                        segIndex++;
                  }
                  else
                        break;
            }

            //    as long as we found a non-TAQ segment
            if (segIndex < numGnSegments) {
                  //    fill up the taqList for the first Name Segment
with
                  //    each of the leading TAQs we found.  If we found
no TAQs above,
                  //    numTempTAQSegs will be 0, so we wont even enter
into the loop.
                  //    Also, since we resticted the loop above, we are
guaranteed to
                  //    not exceed our space for TAQs for a single
segment.
                  for (i = 0; i < numTempTAQSegs; i++)       {
                        gnSegments[segIndex].taqList[i].segString =
gnSegments[i].segString;
                        gnSegments[segIndex].taqList[i].taqAction =
tempTAQList[i]->gnAction;
                        gnSegments[segIndex].taqList[i].taqType =
tempTAQList[i]->taqType;
                        gnSegments[segIndex].numTAQs += 1;
                  }

                  //    now move all the segments back starting with
first name segment
                  //    ousting the leading TAQs.  If we found that the
first segment
                  //    was a name segment, we do not need to move
anything.
                  if (segIndex != 0)       {
                        for (i = segIndex; i < numGnSegments;
i++)   {
                              gnSegments[i - segIndex] = gnSegments[i];
                        }
                        //    note that we now have less segments, since
we removed some segments
                        //    that were TAQ values
                        numGnSegments -= segIndex;
                        //    also, set the segIndex to 0, since we are
now back at the begining
                        segIndex = 0;
                  }

                  //    now start looking at the remaining segments
                  //    along the way, we must keep track of
                  //          -     the index of the last Name segment
we found (start out as 0, since we backed it up to 0)
                  //          -     the index of the last "suffix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //          -     the index of the last "prefix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //
                  //    If we get a:
                  //          Name:
                  //                - associate everything between the
lastNameIndex + 1 and the
                  //                      lastSuffixIndex with
gnSegment[lastNameIndex];
                  //                -     associate everything between
the lastPrefixIndex and
                  //                      segIndex - 1 with this name
segment.
                  //                -     move everything back to oust
the TAQ values from the gnSegment array
                  //                -     mark the new lastNameIndex
(lastNameIndex = segIndex;)
                  //                -     adjust numGnSegments for how
many TAQs we ousted
                  //          "Suffix Like"
                  //                lastPrefixIndex = -
1                 //    previous prefix now considered a suffix
                  //                lastSuffixIndex = segIndex
                  //          "Prefix Like"
                  //                lastPrefixIndex =
segIndex
                  //          End of Segments
                  //                - associate everything between the
lastNameIndex + 1 and segIndex
                                          with gnSegment[lastNameIndex];
                  //                -     adjust numGnSegments for how
many TAQs we had at end
                  //
                  //    Note that we do not do any storing of anything
until we either reach the
                  //    end of the sements, or get a non-taq segment.
                  //
                  //    Also, as we read TAQ segments, we store a
pointer to their retrieved
                  //    structure in a list.  We do this because we must
read ahead before
                  //    we can store a TAQs relevant info (type, action)
as being associated
                  //    with a segment, and we do not want to have to
look up the TAQ info twice.
```

```
                  numTempTAQSegs = 0;
                  lastPrefixIndex = -1;
                  lastSuffixIndex = -1;
                  lastNameIndex = segIndex;
                  segIndex++;               //    look at the next segment
                  while (segIndex < numGnSegments)    {
                        tempTAQRecordPtr = taqTable-
>getTAQSegment(gnSegments[segIndex].segString,

            primaryCultureCode,

            secondaryCultureCode);
                        if (tempTAQRecordPtr == NULL) {
                              //    segment is not a TAQ value

                              //    do an initial check to make sure we
actually got one or more TAQs.
                              //    if not, all we really have to do is
just reflect the new value for
                              //    lastNameIndex.
                              if (numTempTAQSegs > 0) {
                                    //    so associate all taqs between
the previous Name segment and
                                    //    the last suffix with the
previous Name Segment.  Since lastSuffixIndex
                                    //    may be -1 (if there we not
suffixes), we may not even enter this for loop.

                                    //    this variable is necessary
because the segment at lastNameIndex
                                    //    might already have TAQs stored
in its taqList (due to prefixes).
                                    //    We must keep track of where
the next available place in that list is.
                                    nameSegmentTaqListIndex =
gnSegments[lastNameIndex].numTAQs;
                                    tempTAQSegIndex = 0;
                                    for (i = lastNameIndex + 1; (i <=
lastSuffixIndex) && (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT);
i++)  {
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].segString = gnSegments[i].segString;
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>gnAction;
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                          tempTAQSegIndex++;
                                          nameSegmentTaqListIndex++;
                                          gnSegments[lastNameIndex].numT
AQs += 1;
                                    }

                                    //    associate everything at or
past the previous prefix(s) with the name
                                    //    segment we just found.  Again,
since there may not have been any
                                    //    prefixes, we might not even
```

```
enter this for loop
                                    if (lastPrefixIndex != -1)     {
                                          for (i = lastPrefixIndex; (i <
segIndex) && (tempTAQSegIndex < NH_MAX_TAQS_PER_SEGMENT); i++)     {
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].segString = gnSegments[i].segString;
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>gnAction;
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                                tempTAQSegIndex++;
                                                gnSegments[segIndex].num

TAQs += 1;
                                          }
                                    }

                                    //    now move all the segments back
starting with this segment and
                                    //    ending with the last segment.
We move them back to the first
                                    //    segment after the previous
Name segment, which is numTempTAQSegs places
                                    for (i = segIndex; i <
numGnSegments; i++)      {
                                          gnSegments[i - numTempTAQSegs]
= gnSegments[i];
                                    }

                                    //for (i = lastNameIndex + 1; i <
numGnSegments; i++)      {
                                    //    gnSegments[i] = gnSegments[i +
numTempTAQSegs];
                                    //}

                                    numGnSegments -=
numTempTAQSegs;         //    we not have less segments, since we got


//    rid of some TAQs
                                    segIndex -=
numTempTAQSegs;                           //    move our pointer back
too
                                    numTempTAQSegs =
0;                                                     //    clear out
the temp segment array
                              }
                              lastNameIndex =
segIndex;                                 //    mark the new
lastNameIndex

                        }
                        else  {
                              if ((tempTAQRecordPtr->taqType == 'P') ||
(tempTAQRecordPtr->taqType == 'T')) {
                                    //    got a prefix or a title
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
                                    //    only set the prefix index if
```

```
we do not have one on record.
                                                //    otherwise, we will only get
the right most prefix in a string
                                                //    of consecutive prefixes.
                                                if (lastPrefixIndex == -1)
                                                      lastPrefixIndex = segIndex;
                                          }
                                          else  {
                                                //    must be a suffix or qualifier
                                                tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                                numTempTAQSegs++;
                                                lastPrefixIndex = -
1;                //    any previous prefixes now considered a suffix
                                                lastSuffixIndex = segIndex;
                                          }
                                    }
                                    segIndex++;             //    look at next
segment
                  }

                  //    now we are at the end of all segments, so make
sure that any
                  //    TAQs that were trailing get associated with the
last name segment.

                  //    do an initial check to make sure we actually got
one or more TAQs.
                  //    if not, all we really have to do is just reflect
the new value for
                  //    lastNameIndex.
                  if (numTempTAQSegs > 0) {
                        //    associate all the stored taqs with the
last name segment.
                        //    in the loop below:
                        //          i is the index into the gnSegments
list for the TAQ string we are copying
                        //          tempTAQSegIndex is the index into
the tempTAQList for the saved TAQ info
                        //          lastNameIndex is the index into the
gnSegments for the name getting
                        //                the TAQs associated with it.
                        //          gnSegmentTaqListIndex is the index
into the taqList for the name getting
                        //                the TAQs associated with it.
                        //
                        //    We must be careful that we do not
overwrite any TAQs already associated with
                        //    the name (from prefixes).  For this
reason, we use separate indexes for the
                        //    tempTAQList and the gnSegments' taqList.

                        nameSegmentTaqListIndex =
gnSegments[lastNameIndex].numTAQs;
                        tempTAQSegIndex = 0;
                        for (i = lastNameIndex + 1; (i < numGnSegments)
&& (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT); i++)      {
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].segString = gnSegments[i].segString;
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]->gnAction;
```

```
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]->taqType;
                              tempTAQSegIndex++;
                              nameSegmentTaqListIndex++;
                              gnSegments[lastNameIndex].numTAQs += 1;
                        }

                        //    now we can just chop off all the TAQ
segments by reducing numGnSegments.
                        numGnSegments -= numTempTAQSegs;
                  }

            }
            else  {
                  //    we did not get any Non-TAQ segments.  Move all
the segments to the TAQ
                  //    list for the first segment, create a single
segment, and set its string
                  //    value to "".
                  gnSegments[0].numTAQs = 0;    //    set this in case
there were no TAQs (empty string)

                              //    In that case, we would not have
cleared it out orignally
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        gnSegments[0].taqList[i].segString =
gnSegments[i].segString;
                        gnSegments[0].taqList[i].taqAction =
tempTAQList[i]->gnAction;
                        gnSegments[0].taqList[i].taqType =
tempTAQList[i]->taqType;
                        gnSegments[0].numTAQs += 1;
                  }
                  numGnSegments = 1;
                  gnSegments[0].segString = "";
                  gnSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
            }
      }

      //    as a last step, we must make sure that the number of
gnSegments is
      //    now no greater than NH_MAX_SEGS_AFTER_TAQ.  We just ignore
any segments
      //    after the max.
      if (numGnSegments > NH_MAX_SEGS_AFTER_TAQ)
            numGnSegments = NH_MAX_SEGS_AFTER_TAQ;


      //    clear out the TAQ counts for each segment.
      //    This is important because the TAQ segments are not
initalized
      //    if they are not filled in.
      for (i = 0; i < numSnSegments; i++)
            snSegments[i].numTAQs = 0;

      //    Now do the SN segments
      if    (nameParms->getSeparateGnTaqs() == true)  {
            //    init some variables
            segIndex = 0;
            numTempTAQSegs = 0;
            //    Start out by looking for TAQs at the start of the name
field,
            //    before any name segments.
            //    while there are TAQ values at the start of the sn
            //    get their associated TAQ record and place that in
            //    a temporary list.
            while (segIndex < numSnSegments)    {
                  tempTAQRecordPtr = taqTable-
>getTAQSegment(snSegments[segIndex].segString,


      primaryCultureCode,


      secondaryCultureCode);
                  if (tempTAQRecordPtr != NULL) {
                        //    make sure we are not past our space for
TAQs in the temp list
                        //    This would happen if a name field started
out with tons of TAQs
                        if (segIndex < NH_MAX_TAQS_PER_SEGMENT)    {
                              tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                              numTempTAQSegs++;
                        }
                        segIndex++;
                  }
                  else
                        break;
            }

            //    as long as we found a non-TAQ segment
            if (segIndex < numSnSegments) {
                  //    fill up the taqList for the first Name Segment
with
                  //    each of the leading TAQs we found.  If we found
no TAQs above,
                  //    numTempTAQSegs will be 0, so we wont even enter
into the loop.
                  //    Also, since we resticted the loop above, we are
guaranteed to
                  //    not exceed our space for TAQs for a single
segment.
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        snSegments[segIndex].taqList[i].segString =
snSegments[i].segString;
                        snSegments[segIndex].taqList[i].taqAction =
tempTAQList[i]->snAction;
                        snSegments[segIndex].taqList[i].taqType =
tempTAQList[i]->taqType;
                        snSegments[segIndex].numTAQs += 1;
                  }

                  //    now move all the segments back starting with
first name segment
                  //    ousting the leading TAQs.  If we found that the
first segment
                  //    was a name segment, we do not need to move
anything.
                  if (segIndex != 0)       {
                        for (i = segIndex; i < numSnSegments;
i++)  {
                              snSegments[i - segIndex] = snSegments[i];
                        }
                        //    note that we now have less segments, since
we removed some segments
                        //    that were TAQ values
                        numSnSegments -= segIndex;

                        //    also, set the segIndex to 0, since we are
now back at the begining
                        segIndex = 0;
                  }

                  //    now start looking at the remaining segments
                  //    along the way, we must keep track of
                  //          -     the index of the last Name segment
we found (start out as 0, since we backed it up to 0)
                  //          -     the index of the last "suffix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //          -     the index of the last "prefix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //
                  //    If we get a:
                  //          Name:
                  //                - associate everything between the
lastNameIndex + 1 and the
                  //                      lastSuffixIndex with
snSegment[lastNameIndex];
                  //                -     associate everything between
the lastPrefixIndex and
                  //                      segIndex - 1 with this name
segment.
                  //                -     move everything back to oust
the TAQ values from the snSegment array
                  //                -     mark the new lastNameIndex
(lastNameIndex = segIndex;)
                  //                -     adjust numSnSegments for how
many TAQs we ousted
                  //          "Suffix Like"
                  //                lastPrefixIndex = -
1                 //    previous prefix now considered a suffix
                  //                lastSuffixIndex = segIndex
                  //          "Prefix Like"
                  //                lastPrefixIndex =
segIndex
                  //          End of Segments
                  //                - associate everything between the
lastNameIndex + 1 and segIndex
                                          with snSegment[lastNameIndex];
                  //                -     adjust numSnSegments for how
many TAQs we had at end
                  //
                  //    Note that we do not do any storing of anything
until we either reach the
                  //    end of the sements, or get a non-taq segment.
                  //
                  //    Also, as we read TAQ segments, we store a
pointer to their retrieved
                  //    structure in a list.  We do this because we must
```

```
read ahead before
                  //    we can store a TAQs relevant info (type, action)
as being associated
                  //    with a segment, and we do not want to have to
look up the TAQ info twice.


                  numTempTAQSegs = 0;
                  lastPrefixIndex = -1;
                  lastSuffixIndex = -1;
                  lastNameIndex = segIndex;
                  segIndex++;                //    look at the next segment
                  while (segIndex < numSnSegments)    {
                        tempTAQRecordPtr = taqTable-
>getTAQSegment(snSegments[segIndex].segString,

            primaryCultureCode,

            secondaryCultureCode);
                        if (tempTAQRecordPtr == NULL) {
                              //    segment is not a TAQ value

                              //    do an initial check to make sure we
actually got one or more TAQs.
                              //    if not, all we really have to do is
just reflect the new value for
                              //    lastNameIndex.
                              if (numTempTAQSegs > 0) {
                                    //    so associate all taqs between
the previous Name segment and
                                    //    the last suffix with the
previous Name Segment.  Since lastSuffixIndex
                                    //    may be -1 (if there we not
suffixes), we may not even enter this for loop.

                                    //    this variable is necessary
because the segment at lastNameIndex
                                    //    might already have TAQs stored
in its taqList (due to prefixes).
                                    //    We must keep track of where
the next available place in that list is.
                                    nameSegmentTaqListIndex =
snSegments[lastNameIndex].numTAQs;
                                    tempTAQSegIndex = 0;
                                    for (i = lastNameIndex + 1; (i <=
lastSuffixIndex) && (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT);
i++)  {
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].segString = snSegments[i].segString;
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>snAction;
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                          tempTAQSegIndex++;
                                          nameSegmentTaqListIndex++;
                                          snSegments[lastNameIndex].numT
AQs += 1;
```

```
                                    }

                                    //    associate everything at or
past the previous prefix(s) with the name
                                    //    segment we just found.  Again,
since there may not have been any
                                    //    prefixes, we might not even
enter this for loop
                                    if (lastPrefixIndex != -1)     {
                                          for (i = lastPrefixIndex; (i <
segIndex) && (tempTAQSegIndex < NH_MAX_TAQS_PER_SEGMENT); i++)     {
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].segString = snSegments[i].segString;
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>snAction;
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                                tempTAQSegIndex++;
                                                snSegments[segIndex].num
TAQs += 1;
                                          }
                                    }

                                    //    now move all the segments back
starting with this segment and
                                    //    ending with the last segment.
We move them back to the first
                                    //    segment after the previous
Name segment, which is numTempTAQSegs places
                                    for (i = segIndex; i <
numSnSegments; i++)     {
                                          snSegments[i - numTempTAQSegs]
= snSegments[i];
                                    }

                                    numSnSegments -=
numTempTAQSegs;         //    we not have less segments, since we got

//    rid of some TAQs
                                    segIndex -=
numTempTAQSegs;                           //    move our pointer back
too
                                    numTempTAQSegs =
0;                                                    //    clear out
the temp segment array
                              }
                              lastNameIndex =
segIndex;                                 //    mark the new
lastNameIndex

                        }
                        else  {
                              if ((tempTAQRecordPtr->taqType == 'P') ||
(tempTAQRecordPtr->taqType == 'T')) {
                                    //    got a prefix or a title
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
                                    //    only set the prefix index if
we do not have one on record.
                                    //    otherwise, we will only get
the right most prefix in a string
                                    //    of consecutive prefixes.
                                    if (lastPrefixIndex == -1)
                                          lastPrefixIndex = segIndex;
                              }
                              else  {
                                    //    must be a suffix or qualifier
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
                                    lastPrefixIndex = -
1;                //    any previous prefixes now considered a suffix
                                    lastSuffixIndex = segIndex;
                              }
                        }
                        segIndex++;             //    look at next
segment
                  }

                  //    now we are at the end of all segments, so make
sure that any
                  //    TAQs that were trailing get associated with the
last name segment.

                  //    do an initial check to make sure we actually got
one or more TAQs.
                  //    if not, all we really have to do is just reflect
the new value for
                  //    lastNameIndex.
                  if (numTempTAQSegs > 0) {
                        //    associate all the stored taqs with the
last name segment.
                        //    in the loop below:
                        //          i is the index into the snSegments
list for the TAQ string we are copying
                        //          tempTAQSegIndex is the index into
the tempTAQList for the saved TAQ info
                        //          lastNameIndex is the index into the
snSegments for the name getting
                        //                the TAQs associated with it.
                        //          snSegmentTaqListIndex is the index
into the taqList for the name getting
                        //                the TAQs associated with it.
                        //
                        //    We must be careful that we do not
overwrite any TAQs already associated with
                        //    the name (from prefixes).  For this
reason, we use separate indexes for the
                        //    tempTAQList and the snSegments' taqList.

                        nameSegmentTaqListIndex =
snSegments[lastNameIndex].numTAQs;
                        tempTAQSegIndex = 0;
                        for (i = lastNameIndex + 1; (i < numSnSegments)
&& (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT); i++)      {
                              snSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].segString = snSegments[i].segString;
                              snSegments[lastNameIndex].taqList[nameSegm
```

```
entTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]->snAction;
                              snSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]->taqType;
                              tempTAQSegIndex++;
                              nameSegmentTaqListIndex++;
                              snSegments[lastNameIndex].numTAQs += 1;
                        }

                        //    now we can just chop off all the TAQ
segments by reducing numSnSegments.
                        numSnSegments -= numTempTAQSegs;
                  }

            }
            else  {
                  //    we did not get any Non-TAQ segments.  Move all
the segments to the TAQ
                  //    list for the first segment, create a single
segment, and set its string
                  //    value to "".
                  snSegments[0].numTAQs = 0;    //    set this in case
there were no TAQs (empty string)

                              //    In that case, we would not have
cleared it out orignally
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        snSegments[0].taqList[i].segString =
snSegments[i].segString;
                        snSegments[0].taqList[i].taqAction =
tempTAQList[i]->snAction;
                        snSegments[0].taqList[i].taqType =
tempTAQList[i]->taqType;
                        snSegments[0].numTAQs += 1;
                  }
                  numSnSegments = 1;
                  snSegments[0].segString = "";
                  snSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
            }
      }

      //    as a last step, we must make sure that the number of
gnSegments is
      //    now no greater than NH_MAX_SEGS_AFTER_TAQ.  We just ignore
any segments
      //    after the max.
      if (numSnSegments > NH_MAX_SEGS_AFTER_TAQ)
            numSnSegments = NH_MAX_SEGS_AFTER_TAQ;

}



//    function to generate index keys for this name.
//    Each key includes a portion for the GN and a portion
//    for the SN.
//    We currently support two key lengths, 32 bits or 64 bits.
//    The GN length does not have to be the same as the SN length,
//    but GN keys generated must be the same length (similarly for
//    SN).  Thus the full key length could be:
//
//          64:         Both GN and SN are 32 bits
//          96:         Gn is 64, but SN is 32
//          96:         Gn is 32, but SN is 64
//          128:  Both GN and SN are 64 bits
//
//    Keys are generated by name stem segment.  The first key
//    consists of a key for the first GN segment, and a key
//    for the first SN segment.  The second key
//    consists of a key for the second GN segment, and a key
//    for the second SN segment.  When there are a differing number
//    of GN and SN segments, the final segment of the name
//    field with the fewer number of segments is repeated.
//    Thus, the number of keys generated is given by the formula:
//                max(numGnSegs, numSnSegs)
//
//    We do things this way so that a name has the same number of keys
//    for both GN and SN, and in fact we can view the two keys as one
//    contiguous key that can be passed to comparison functions as a
//    single value.
//
//    Note that we are talking about stem segments (TAQ segments have
//    been removed).
//
//    maxKeys specifies how many keys the caller can fit into
//    keyBuff.  It is up to the caller to make sure that they have
allocated
//    enough space in the keyBuff to hold maxKeys.
unsigned    char  NHNameData::genIndexKeys(int maxKeys, NHKeyWidth
gnKeyWidth,

                  NHKeyWidth snKeyWidth, void *keyBuff)
{
      int   numKeysGenerated = 0;
      int gnSegIndex = 0;
      int snSegIndex = 0;
      unsigned    int   *keyPtr = (unsigned     int   *)keyBuff;


      while (numKeysGenerated < maxKeys)   {
            if ((gnSegIndex >= numGnSegments) && (snSegIndex >=
numSnSegments))
                  break;
            else  {
                  numKeysGenerated++;
                  //    make sure that if one segment is now at the end,
                  //    we stay on the last segment
                  if (gnSegIndex == numGnSegments)
                        gnSegIndex--;
                  if (snSegIndex == numSnSegments)
                        snSegIndex--;

                  if (gnKeyWidth == NH_KEY_WIDTH_32)   {
                        //    gn key length is 32
                        *keyPtr =
globalDigraphBitmapArray.get32BitKeyForToken(gnSegments[gnSegIndex].segS
tring);

                        keyPtr++;         //    move the pointer by 4
bytes
                  }
                  else  {
                        //    gn key length is 64
                        globalDigraphBitmapArray.get64BitKeyForToken(gnS
egments[gnSegIndex].segString,


            (bit_64_t *)keyPtr);
                        keyPtr += 2;            //     move the pointer
by 8 bytes
                  }

                  if (snKeyWidth == NH_KEY_WIDTH_32)    {
                        //     gn key length is 32
                        *keyPtr =
globalDigraphBitmapArray.get32BitKeyForToken(snSegments[snSegIndex].segS
tring);
                        keyPtr++;         //     move the pointer by 4
bytes
                  }
                  else  {
                        //     gn key length is 64
                        globalDigraphBitmapArray.get64BitKeyForToken(snS
egments[snSegIndex].segString,


            (bit_64_t *)keyPtr);
                        keyPtr += 2;            //     move the pointer
by 8 bytes
                  }

                  //     advance the segment indexes
                  snSegIndex++;
                  gnSegIndex++;
            }
      }
      return numKeysGenerated;
}
```

```
//    File:  NHEvalNameData.cpp
//
//    Description:
//
//          Implementation to the NHEvalNameData class.
//
//
//    History:
//
//          5/14/97           EFB         Created
//          9/1/97            EFB         Lots of changes to support
retaining segment scores in
//                                                          best mode so
that sorting can be more detailed and accurate
//          10/31/97    EFB         Made several member functions
protected, and made performComp()
//                                                          a friend of
NHQueryNameData.  Also changed performComp to
//                                                          NOT delete
objects that are not passed on to the resultslist,
//                                                          to
accomodate the new method of deleting NHEvalNameData objects.
//          11/03/97    EFB         Added a new function,
calcNameScore() and made it virtual.
//                                                          removed
virtual from performComp.  The perform comp method
//                                                          was too
complicated to be subclassed.  We really only want
//                                                          callers to
be able to affect the name score and the determination
//                                                          of
HIT/NO_HIT.  These are now the only virtual functions.  Both
//                                                          are now
inline in the header file so the caller knows exactly
//                                                          what is
happening in these functions if they decide to subclass
//                                                          and
override. OOPS, I forgot compareScore(), which is also
//                                                          virtual - we
want them to be able to change how hits are sorted.
//
//          3/02/98     EFB               Made lots of changes necessary
when I moved a bunch of
//                                                          parameters
(the ones associated with parsing the name)
//                                                          from the
NHCompParms class into a new class called NHNameParms.
//                                                          and renamed
the NHCompParms class to NHCompParms.
//          3/20/98     EFB         Changed names to NH from SN


#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include    "NHEvalNameData.hpp"
#include    "NHQueryNameData.hpp"
#include    "NH_util.hpp"
#include    "NH_queens_arrays.hpp"
```

```
#include    "NHVariantTable.hpp"
#include    "NHResultsList.hpp"
#include    "NHTAQTable.hpp"
#include    "NHNameParms.hpp"


//    private, non-member function prototype
static double     NH_digraph_score(char *qSeg, int qSegLen,

char *evalSeg, int evalSegLen,
                          bool useLeftDigraphBias);

static      double      NH_best_score(int numQSegs, int numEvalSegs,
NHSegScoreMode scoreMode,
                                                                   double
scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ]);

void  NH_best_score_for_highest_mode(int xDim, int yDim, double
highestScore,
                                                                   double
*bestSegScores,
                                                                   double
scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ]);

static      double NH_calc_score(   SegList qSegs, int numQSegs,
                                                                   SegLis
t evalSegs, int numEvalSegs,
                                                                   SegLis
tVariants querySegmentVariants,
                                                                   char
                                     *primaryCulture,
                                                                   char
                                     *secondaryCulture,
                                                                   NHComp
Parms *compParms,
                                                                   NHName
Parms *nameParms,
                                                                   NHName
Fields nameField,
                                                                   char
*origQNameField,
                                                                   char
*origEvalNameField,
                                                                   int
*numSegsScored,
                                                                   double
*bestSegScores);

static      void  NH_apply_TAQs_to_score(double *diScore, Segment *qSeg,
Segment *evalSeg,
                         double absDelTAQFactor,

                         double absDisTAQFactor,

                         double delTAQFactor,

                         double disTAQFactor);

static      bool NH_check_compressed_name(char *qSegString, char
*evalSegString,
                                        char *compressCharsPart1,
                                        char  *compressCharsPart2);

NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *aGn, char
*aSn) :
                                                  NHNameData(nParms, aGn,
aSn)
{
      resetScores();
}


NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *aGn, char
*aSn, char *aMn) :
                                                  NHNameData(nParms, aGn,
aSn, aMn)
{
      resetScores();
}


NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *name,
NHNameFormat nameFormat)      :
                                                  NHNameData(nParms, name,
nameFormat)
{
      resetScores();
}


//    constuct an object from an archived representation in
//    a stream.
//
//    The archive is in the following order
//
//    gnLen
//    snLen
//    nameStorage
NHEvalNameData::NHEvalNameData(NHNameParms *nParms, istream &inStream) :
                                                  NHNameData(nParms,
inStream)
{
      //    read the gn, sn and name scores
      if (inStream)
            inStream.read((char *)&gnScore, sizeof(gnScore));
      if (inStream)
            inStream.read((char *)&snScore, sizeof(snScore));
      if (inStream)
            inStream.read((char *)&nameScore, sizeof(nameScore));

      //    seg differentials
      if (inStream)
            inStream.read((char *)&gnSegDifferential,
sizeof(gnSegDifferential));
      if (inStream)
            inStream.read((char *)&snSegDifferential,
sizeof(snSegDifferential));
```

```
      //    read the number of gn segs scored, and however many scores
we need     inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
      if (inStream)
            inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
      if (inStream)     {
            if (numGnSegsScored > 0)       {
                  inStream.read((char *)gnSegScores, numGnSegsScored *
sizeof(double));
            }
      }

      //    read the number of sn segs scored, and however many scores
we need
      if (inStream)
            inStream.read((char *)&numSnSegsScored,
sizeof(numSnSegsScored));
      if (inStream)     {
            if (numSnSegsScored > 0)       {
                  inStream.read((char *)snSegScores, numSnSegsScored *
sizeof(double));
            }
      }
}


NHEvalNameData::~NHEvalNameData()
{
}


bool  NHEvalNameData::archiveData(ostream &outStream)
{
      bool  rc = true;


      rc = NHNameData::archiveData(outStream);
      if (rc)     {
            //    read the gn, sn and name scores
            outStream.write((char *)&gnScore, sizeof(gnScore));
            outStream.write((char *)&snScore, sizeof(snScore));
            outStream.write((char *)&nameScore, sizeof(nameScore));

            //    seg differentials
            outStream.write((char *)&gnSegDifferential,
sizeof(gnSegDifferential));
            outStream.write((char *)&snSegDifferential,
sizeof(snSegDifferential));

            //    read the number of gn segs scored, and however many
scores we need    inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
            outStream.write((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
            if (numGnSegsScored > 0)   ,   {
                  outStream.write((char *)gnSegScores, numGnSegsScored *
sizeof(double));
            }
```

```
          //    read the number of sn segs scored, and however many
scores we need
          outStream.write((char *)&numSnSegsScored,
sizeof(numSnSegsScored));
          if (numSnSegsScored > 0)      {
               outStream.write((char *)snSegScores, numSnSegsScored *
sizeof(double));
          }
     }

     return rc;
}


//    note that this function is a friend of NHQueryNameData, which is
//    why we are able to access private member functions of that class.
void  inline NHEvalNameData::calcComponentScores(NHQueryNameData
*queryName)
{
     char                               *primaryCulture = nameParms-
>primaryCultureCode;
     char                               *secondaryCulture = nameParms-
>secondaryCultureCode;

     //    do the digraph compare and set the scores
     gnScore = NH_calc_score(queryName->gnSegments, queryName-
>numGnSegments,
                                                                      gnSegm
ents, numGnSegments,
                                                                      queryN
ame->gnSegmentVariants,
                                                                      primar
yCulture, secondaryCulture,
                                                                      compPa
rms,
                                                                      namePa
rms,
                                                                      NH_FIR
ST_NAME,
                                                                      queryN
ame->gn, gn,
                                                                      &numGn
SegsScored,
                                                                      gnSegS
cores);
     snScore = NH_calc_score(queryName->snSegments, queryName-
>numSnSegments,
                                                                      snSegm
ents, numSnSegments,
                                                                      queryN
ame->snSegmentVariants,
                                                                      primar
yCulture, secondaryCulture,
                                                                      compPa
rms,
                                                                      namePa
rms,
                                                                      NH_LAS
T_NAME,
                                                                      queryN
ame->sn, sn,
                                                                 &numSn
SegsScored,
                                                                 snSegS
cores);
}


//    note that this function is a friend of NHQueryNameData, which is
//    why we are able to access private member functions of that class.
NHReturnCode      NHEvalNameData::performComp(NHQueryNameData
*queryName,

                                                           NHCompParms
*someCompParms)
{
      NHReturnCode            compResult;
      NHResultsList           *resultList;


      //    save the compParms so that they can be easily referenced
      //    throughout the comparison process.
      compParms = someCompParms;

      calcComponentScores(queryName);

      //    call a method to calculate the name score.
      calcNameScore();

      //    store the segments differentials, in case we get a tie
score.
      gnSegDifferential = abs(numGnSegments - queryName-
>getNumGnSegments());
      snSegDifferential = abs(numSnSegments - queryName-
>getNumSnSegments());

      //    Now call the getCompResult() function to get the return
value
      //    (i.e. was it a match?)
      compResult = getCompResult();

      //    now see if we are working with a results list
      resultList = queryName->getResultsList();
      if (resultList != NULL) {
            //    we are using a result list.  If this is a hit, add it
            //    to the result list.
            //    Otherwise, delete it
            if (compResult == NH_MATCH)   {
                  NHReturnCode            tempInsertResult;

                  //    make sure the insert works.  If so, don't mess
with
                  //    the compResult, so the comparison will be
returned
                  //    as a hit.  If there was an error, delete this
object,
                  //    and save the error code so it can be returned.
                  tempInsertResult = resultList->addHit(this);
                  if (tempInsertResult != NH_SUCCESS) {
                        compResult = tempInsertResult;
                  }
            }
      }
      return compResult;
}


//    used only when the segment mode is set to HIGHEST.
//    It compares the segment scores the were retained when
//    the name was compared to the query name.
//    We are comparing the segment scores for two (pre-scored)
//    eval names.  The comparison should find which name has
//    the "best" set of segment scores, where best is defined
//    as "the one with the highest best score".  If the best
//    score results in a tie, we move on to the second best score,
//    and so on until we find a difference, or there are no more
//    segments to compare.  Each name has variables numGnSegsScored
//    and numSnSegsScored, that tell how many segments were scored
//    in the name.  We do up to N comparisons, where N is the larger
//    of the number of segments scored in each name.  Where one name
//    has less segments scored than the other, a default value of
//    NH_DEFAULT_MISSING_SEGMENT_SCORE is assigned.  This is so that
//    a scored segment has to beat some threshold to be considered
//    better than nothing at all.
//
double      NHEvalNameData::compareSegmentScores(NHEvalNameData
*scoredName, NHNameFields nameField)
{
      double      scoreDiff;
      int               maxComparisons;
      double      *thisEvalScores;
      double      *compEvalScores;
      int               numSegsScoredForThisEval;
      int               numSegsScoredForCompEval;


      if (nameField == NH_LAST_NAME)      {
            thisEvalScores = snSegScores;
            compEvalScores = scoredName->snSegScores;
            numSegsScoredForThisEval = numSnSegsScored;
            numSegsScoredForCompEval = scoredName->numSnSegsScored;
      }
      else  {
            thisEvalScores = gnSegScores;
            compEvalScores = scoredName->gnSegScores;
            numSegsScoredForThisEval = numGnSegsScored;
            numSegsScoredForCompEval = scoredName->numGnSegsScored;
      }
      maxComparisons = numSegsScoredForThisEval >
numSegsScoredForCompEval ? numSegsScoredForThisEval :
numSegsScoredForCompEval;

      for (int i = 0; i < maxComparisons; i++)  {
            if (i >= numSegsScoredForThisEval)
                  thisEvalScores[i] = NH_DEFAULT_MISSING_SEGMENT_SCORE;
            else        //    we can do an else because only one segment
can be missing, not both
                  if (i >= numSegsScoredForCompEval)
                        compEvalScores[i] =
NH_DEFAULT_MISSING_SEGMENT_SCORE;

            scoreDiff = compEvalScores[i] - thisEvalScores[i];
            if (scoreDiff != 0)
```

```
                    break;
            }

            return scoreDiff;
}

/*************************************************************************
****/

/* NH_calc_score
   Performs a string comparison on two name fields.
       Returns a value between 0.00 and
   1.00, with 1.00 being an exact-fit

*/
double NH_calc_score(   SegList qSegs, int numQSegs,
                                                                  SegLis
t evalSegs, int numEvalSegs,
                                                                  SegLis
tVariants querySegmentVariants,
                                                                  char
                                    *primaryCulture,
                                                                  char
                                    *secondaryCulture,
                                                                  NHComp
Parms *compParms,
                                                                  NHName
Parms *nameParms,
                                                                  NHName
Fields nameField,
                                                                  char
*origQNameField,
                                                                  char
*origEvalNameField,
                                                                  int
*numSegsScored,
                                                                  double
*bestSegScores)
{
      NHAnchorSegMode         anchorSeg;
      NHSegScoreMode          scoreMode;
      double                                      oopsFactor;
      double                                      absDelTAQFactor;
      double                                      absDisTAQFactor;
      double                                      delTAQFactor;
      double                                      disTAQFactor;
      bool                                        matchInit;
      double                                      initScore;
      double                                      initialOnInitialMatchSco
re;
      bool                                        checkVariant;
      //    double                                      variantScore;
      bool                                        leftDigraphBias;
      double                                      anchorFactor;
      double                                      nameUnknownScore;
      double                                      noNameScore;
   double                                   scoresTable[NH_MAX_SEGS_AFTER_
TAQ][NH_MAX_SEGS_AFTER_TAQ];  //    scores for segment pairs
      int                                               qIndex;
      //    temp index for query segments
```

```
     int                                                         evalIndex;  //
temp index for eval segments
     int                                                         qSegLen;
//    hold string length of query segment
     int                                                         evalSegLen; //
hold string length of eval segment
  double                                              diScore =
0.0;         //    temp score for single pair comparison
     double                                                 hiScore =
0.0;         //    temp score to hold best score as we iterate,

                                                      //    which lets us avoid
best_score in mode=BEST
     bool                                                   areVariants;
//    temp flag to hold if the pair are variants
  double                                                      returnValue = 0.0;
     NHVariantTable                                    *variantTable;
     double                                                          varScore;
     NHVarId                                                         evalSegVarId
;
     bool                                                   scoreTaqs;
     double                                                 compressedNameScore;
     bool                                                   checkCompressedName;


     //    set some paramters based on the name field
  if (nameField == NH_LAST_NAME) {
    anchorSeg = compParms->getSnAnchorSegmentMode();
    scoreMode = compParms->getSnSegmentScoreMode();
    oopsFactor = compParms->getSnOOPSFactor();
    matchInit = compParms->getMatchSnIntial();
    initScore = compParms->getSnInitialScore();
            initialOnInitialMatchScore = compParms-
>getSnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseSnVariants();
            anchorFactor = compParms->getSnAnchorFactor();
            leftDigraphBias = compParms->getUseSnLeftBias();
            nameUnknownScore = compParms->getLNUScore();
            noNameScore = compParms->getNLNScore();
            scoreTaqs = compParms->getScoreSnTAQs();
            absDelTAQFactor = compParms->getAbsDelSnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisSnTAQFactor();
            delTAQFactor = compParms->getDelSnTAQFactor();
            disTAQFactor = compParms->getDisSnTAQFactor();
            compressedNameScore = compParms->getSnCompressedNameScore();
            checkCompressedName = compParms->getCheckSnCompressedName();
            variantTable = nameParms->snVariantTable;
     }
  else {
    anchorSeg = compParms->getGnAnchorSegmentMode();
    scoreMode = compParms->getGnSegmentScoreMode();
    oopsFactor = compParms->getGnOOPSFactor();
    matchInit = compParms->getMatchGnIntial();
    initScore = compParms->getGnInitialScore();
            initialOnInitialMatchScore = compParms-
>getGnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseGnVariants();
            anchorFactor = compParms->getGnAnchorFactor();
            leftDigraphBias = compParms->getUseGnLeftBias();
            nameUnknownScore = compParms->getFNUScore();
            noNameScore = compParms->getNFNScore();
            scoreTaqs = compParms->getScoreGnTAQs();
            absDelTAQFactor = compParms->getAbsDelGnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisGnTAQFactor();
            delTAQFactor = compParms->getDelGnTAQFactor();
            disTAQFactor = compParms->getDisGnTAQFactor();
            compressedNameScore = compParms->getGnCompressedNameScore();
            checkCompressedName = compParms->getCheckGnCompressedName();
            variantTable = nameParms->gnVariantTable;
  }

      //    clear out the scores table
  for (qIndex = 0; qIndex < NH_MAX_SEGS_AFTER_TAQ; ++qIndex)
    for (evalIndex = 0; evalIndex < NH_MAX_SEGS_AFTER_TAQ; ++evalIndex)
      scoresTable[qIndex][evalIndex] = 0.0;

      //    now go through each possible combination of segment pairs
      //    (created by matching a query segment against an eval
segment).
      //    Store the scores in the scoresTable.
  for (qIndex = 0; qIndex < numQSegs; ++qIndex) {

    qSegLen = strlen(qSegs[qIndex].segString);

    for (evalIndex = 0; evalIndex < numEvalSegs; ++evalIndex) {

      evalSegLen = strlen(evalSegs[evalIndex].segString);

                  //    first check for either the query or eval segment
being
                  //    blank.
                  if ((qSegLen == 0) || (evalSegLen == 0))  {
                        //    We make a distinction between "unknown"
                        //    and "none".  The table below shows the
scores
                        //    we assign for the various combinations of
Known - K,
                        //    Unknown - U, and None -N.
                        //
                        //          |     K
      |     U
       |    N
                        //    -----------------------------------------
-------------------------
                        //    K     |     N/A
|     unknownScore                                |    NoneScore
                        //    -----------------------------------------
-------------------------
                        //    U     |     unknownScore|     (unknownScor
e + 1) / 2 |      (unknownScore + 1) / 2
                        //    -----------------------------------------
-------------------------
                        //    N     |     NoneScore         |     (unkno
wnScore + 1) / 2 |      (NoneScore + 1) / 2
                        //    -----------------------------------------
-------------------------


                        if (qSegs[qIndex].status ==
NH_NAME_FIELD_STATUS_KNOWN)   {
                              //    we should not need to check for both
being known
```

```
//      File:  NHQueryNameData.cpp
//
//      Description:
//
//             Implementation to the NHQueryNameData class.
//
//
//      History:
//
//             5/14/97      EFB          Created
//             3/20/98      EFB          Changed names to NH from SN
//


#include <string.h>
#include <stdio.h>


#include      "NHQueryNameData.hpp"
#include      "NHVariantTable.hpp"
#include      "NHResultsList.hpp"
#include      "NH_util.hpp"
#include      "NHDigraphBitmapArray.hpp"
#include      "NHNameParms.hpp"


extern        NHDigraphBitmapArray     globalDigraphBitmapArray;

#define       NH_INDEX_THRESH    0.5


NHQueryNameData::NHQueryNameData(NHNameParms *nParms, char *aGn, char
*aSn) :
                                                     NHNameData(nParms, aGn,
aSn)
{
      resultsList = NULL;
      keysArray = NULL;
      numBitsInGnKeys = NULL;
      numBitsInSnKeys = NULL;
      processVariantValues(nParms->gnVariantTable,

nParms->snVariantTable);
}


NHQueryNameData::NHQueryNameData(NHNameParms *nParms, char *aGn, char
*aSn, char *aMn) :
                                                     NHNameData(nParms, aGn,
aSn, aMn)
{
      resultsList = NULL;
      keysArray = NULL;
      numBitsInGnKeys = NULL;
      numBitsInSnKeys = NULL;
      processVariantValues(nParms->gnVariantTable,

nParms->snVariantTable);
}
```

```
NHQueryNameData::NHQueryNameData(NHNameParms *nParms, char *name,
NHNameFormat nameFormat)        :
                                                NHNameData(nParms, name,
nameFormat)
{
      resultsList = NULL;
      keysArray = NULL;
      numBitsInGnKeys = NULL;
      numBitsInSnKeys = NULL;
      processVariantValues(nParms->gnVariantTable,

nParms->snVariantTable);
}


NHQueryNameData::~NHQueryNameData()
{
      if (keysArray != NULL)
            delete [] keysArray;
      if (numBitsInGnKeys != NULL)
            delete [] numBitsInGnKeys;
      if (numBitsInSnKeys != NULL)
            delete [] numBitsInSnKeys;
}


//    Function to get a pointer to a NHVariant object for each name
//    segment.  We do this here, in the query
//    name, so that lookups only have to be done once for the query
name.
//    Note also that we check first to make sure that we are supposed to
be
      using variants (we do this per name field).
void  NHQueryNameData::processVariantValues(NHVariantTable
*gnVariantTable,

                                                            NHVariantTab
le *snVariantTable)
{
      int         i;


      if (nameParms->getUseGnVariants())  {
            for (i = 0; i < numGnSegments; i++)
                  gnSegmentVariants[i] = gnVariantTable-
>getVariantObjectForName(gnSegments[i].segString);
      }
      if (nameParms->getUseSnVariants())  {
            for (i = 0; i < numSnSegments; i++)
                  snSegmentVariants[i] = snVariantTable-
>getVariantObjectForName(snSegments[i].segString);
      }
}


//    function to allocate space for, and generate, the keys for
//    this query name.  The caller calls this explicitly with the
//    desired key widths for the GN and SN.  We use these
//    values in conjunction with the numGnSegments and numSnSegments
//    to calculate how big to make the array that will hold the keys.
void  NHQueryNameData::prepareKeys(NHKeyWidth gnKeyWidth,

                                         NHKeyWidth snKeyWidth)
{
      int                                 keyArraySize;
      unsigned char      largerNumberOfSegments;
      int                                 fullKeyLen;


      //     first allocate the keys
      if (numSnSegments > numGnSegments)
            largerNumberOfSegments = numSnSegments;
      else
            largerNumberOfSegments = numGnSegments;
      if (gnKeyWidth == NH_KEY_WIDTH_32)   {
            if (snKeyWidth == NH_KEY_WIDTH_32)
                  fullKeyLen = 64;
            else
                  fullKeyLen = 96;
      }
      else  {
            if (snKeyWidth == NH_KEY_WIDTH_32)
                  fullKeyLen = 96;
            else
                  fullKeyLen = 128;
      }
      keyArraySize = largerNumberOfSegments * fullKeyLen;
      keysArray = new unsigned int[keyArraySize];

      //     save the key lengths
      queryGnKeyWidth = gnKeyWidth;
      querySnKeyWidth = snKeyWidth;

      //     now generate the keys for the query
      numBitmapKeys = genIndexKeys(largerNumberOfSegments, gnKeyWidth,

                        snKeyWidth, keysArray);

      //     now allocate space for the arrays that hold the number of
      //     bits turned on for each key in the GN and SN.
      numBitsInGnKeys = new unsigned char[largerNumberOfSegments];
      numBitsInSnKeys = new unsigned char[largerNumberOfSegments];

      unsigned char *keysArrayBytePtr = (unsigned char *)keysArray;
      for (int i = 0; i < numBitmapKeys; i++)    {
            if (gnKeyWidth == NH_KEY_WIDTH_32)   {
                  //     the number of bits turned on is the sum of the
number of bits
                  //     in each of the 4 bytes that make up the 32 bit
value
                  numBitsInGnKeys[i] =
globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePtr++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++));
            }
            else  {
                  //    the number of bits turned on is the sum of the
number of bits
                  //    in each of the 8 bytes that make up the 64 bit
value
                  numBitsInGnKeys[i] =
globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePtr++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++));
            }

            // now do the surname
            if (snKeyWidth == NH_KEY_WIDTH_32)   {
                  //    the number of bits turned on is the sum of the
number of bits
                  //    in each of the 4 bytes that make up the 32 bit
value
                  numBitsInSnKeys[i] =
globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePtr++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++));
            }
            else  {
                  //    the number of bits turned on is the sum of the
number of bits
                  //    in each of the 8 bytes that make up the 64 bit
value
                  numBitsInSnKeys[i] =
globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePtr++)) +

            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++)) +
            globalDigraphBitmapArray.getNumBitsForByte(*(keysArrayBytePt
r++));
            }

      }
}


#define     NH_EITHER_NH_OR_GN      1
#define     NH_BOTH_NH_AND_GN       2

//    function to compare the key(s) for this query name against
//    a supplied key from an eval name.  Before this function is
//    called, the caller must have called the
//    perpareKeys() method, which sets the gnKeyLength and
//    snKeyLength variables, and generates the keys for this
//    query name.
//    The comparison is performed by looking at the givename name
//    and surname portions of the key separately.  For each of these
//    subkeys, we see how many bits match, a calculate the quotient of
//    matching bits / bits that could have matched.  This score is
//    compared to ???. If the score for either the GN or SN comparison
//    is favorable, the function returns true to indicate that the
//    evaluation name associated with the supplied key is a possible
//    match, and should be retrieved for further consideration.
//    Since this object (the query) could generate multiple keys,
//    we may have to perform several comparisons.
bool  NHQueryNameData::compareKey(unsigned int  *evalBitMapKey, unsigned
char numEvalKeys)
{
      bool                                  rc = false;
      unsigned    int         *evalKeyPtr;
      unsigned    int         *queryKeyPtr;
      unsigned    int         *masterQueryKeyPtr = keysArray;
      unsigned    int         maskedVal;
      unsigned    char  numBitsThatMatched;
      unsigned    char  *bytePtr;
      bool                                  passedGn = false;
      bool                                  passedSn = false;
      int                                         indexMode =
NH_BOTH_NH_AND_GN;

      //    for each of the query's keys, do both a SN and GN comparison
      //    out nested loop compares the first GN and SN query key to
      //    all the eval keys (inner loop), and then moves on to the
```

```
next
      //    query key (outter loop).
      for (int i = 0; (i < numBitmapKeys) && (rc == false); i++)  {
            evalKeyPtr = evalBitMapKey;                //    start the
eval ptr at the beggining
            for (int j = 0; j < (int)numEvalKeys; j++)      {

                  //    place the queryKeyPtr back to the beggining of
the
                  //    current query key.  This value gets advanced
after we have
                  //    compared the current query key to all eval keys
                  queryKeyPtr = masterQueryKeyPtr;

                  //    first, check the given name
                  if (queryGnKeyWidth == NH_KEY_WIDTH_32)   {
                        //    just compare a 32 bit key for the gn
                        maskedVal = *evalKeyPtr & *queryKeyPtr;
                        bytePtr = (unsigned     char  *)&maskedVal;
                        numBitsThatMatched =
globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++)) +

                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
)) +

                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
)) +

                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
));

                        if ((double)numBitsThatMatched /
(double)numBitsInGnKeys[i] > NH_INDEX_THRESH)    {
                              if (indexMode ==
NH_EITHER_NH_OR_GN)     {
                                    rc = true;
                                    break;
                              }
                              else  {
//    looking for both, is SN already set?
                                    if
(passedSn)  {            //    yes, so we matched both
                                          rc = true;
                                          break;
                                    }
                                    else
//    no, just set the gn flag
                                          passedGn = true;
                              }
                        }
                        evalKeyPtr++;     //    advance pointers
                        queryKeyPtr++;
                  }
                  else  {
                        //    just compare a 64 bit key for the gn
                        maskedVal = *evalKeyPtr & *queryKeyPtr;
                        bytePtr = (unsigned     char  *)&maskedVal;
                        numBitsThatMatched =
globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++)) +

                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
));
                        evalKeyPtr++;     //    advance pointers to get
to second 32 bits in this 64 bit key
                        queryKeyPtr++;
                        maskedVal = *evalKeyPtr & *queryKeyPtr;
                        bytePtr = (unsigned     char  *)&maskedVal;
                        numBitsThatMatched +=
globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++)) +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
));
                        if ((double)numBitsThatMatched /
(double)numBitsInGnKeys[i] > NH_INDEX_THRESH)   {
                              if (indexMode ==
NH_EITHER_NH_OR_GN)     {
                                    rc = true;
                                    break;
                              }
                              else  {
//    looking for both, is SN already set?
                                    if
(passedSn)  {           //    yes, so we matched both
                                          rc = true;
                                          break;
                                    }
                                    else
//    no, just set the gn flag
                                          passedGn = true;
                              }
                        }
                        evalKeyPtr++;     //    advance pointers
                        queryKeyPtr++;
                  }

                  //    now, check the surname
                  if (querySnKeyWidth == NH_KEY_WIDTH_32)   {
                        //    just compare a 32 bit key for the sn
                        maskedVal = *evalKeyPtr & *queryKeyPtr;
                        bytePtr = (unsigned     char  *)&maskedVal;
                        numBitsThatMatched =
globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++)) +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
                  globalDigraphBitmapArray.getNumBitsForByte(*(bytePtr++
))  +
NH_NAME_FIELD_STATUS_NON_EXISTANT;
                  }     else if     (!strcmp(gnSegments[i].segString,
"MNU")){
                        gnSegments[i].segString[0] = EOS;
                        gnSegments[i].status =
NH_NAME_FIELD_STATUS_UNKNOWN;
                  }
            }
      }

      //    now the sn segs
      if (numSnSegments == 0) {
            numSnSegments = 1;
            snSegments[0].segString = "";
            snSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
      }
      else if (nameParms->getCheckSnUnknowns()) {
            for (i = 0; i < numSnSegments; i++)       {
                  if    (!strcmp(snSegments[i].segString, "NLN")){
                        snSegments[i].segString[0] = EOS;
                        snSegments[i].status =
NH_NAME_FIELD_STATUS_NON_EXISTANT;
                  }     else if     (!strcmp(snSegments[i].segString,
"LNU"))     {
                        snSegments[i].segString[0] = EOS;
                        snSegments[i].status =
NH_NAME_FIELD_STATUS_UNKNOWN;
                  }
            }
      }
}


//    function to go through the segments and for each one, see if
//    it is a TAQ value.  If so, we associate the TAQ with the previous
//    or following segment, depending on its type (i.e. prefix, suffix,
etc).
//    When we store the TAQ, we also store the action associated with
//    the TAQ (currently DELETE or DISREGARD), since this information
//    will be required to determine how to adjust the base segment score
//
//    Deciding which segment to associate a TAQ with can get pretty
//    hairy, especially when mulitple TAQs can be in a name field
//    consecutively.  We use the Following rules for single TAQ values:
//
//    TAQ Type                Segment to Associate with
//
//    Prefix                        next segment
//    Suffix                        previous segment
//    Infix                         Not supported yet
//    Title                         next segment
//    Qualifier               previous segment
//
//    These are the basic rules for figuring out which segment to
associate
//    TAQs with:
//
//    -     Any TAQ segments before the first Name segment are
associated with
//          the first name segment
```

```
//
//      -       Any TAQ segments after the last Name segment are associated
with
//              the last Name segment
//
//      -       For TAQs that are surrounded by Name segments :
//
//              -       All TAQs between a Name segment (on the left) and a
suffix (qualifier)
//                      (on the right) are associated with the Name Segment.
//
//              -       All TAQs not fitting the above are assoicated with the
Name segment
//                      they proceed.
//
void    NHNameData::processTAQValues(NHTAQTable *taqTable)
{
        //      NHTAQAction                     taqAction;
        int                                             i;
        NH_TAQRecordPtr     tempTAQList[NH_MAX_TAQS_PER_SEGMENT];
//      temp list of TAQs found
        int                                             tempTAQSegIndex;    //
temp index for the tempTaqList
        NH_TAQRecordPtr     tempTAQRecordPtr; //      pointer to structure for
a TAQ record
        int                                             numTempTAQSegs;
//      how many TAQs did we find
        int                                             segIndex;
                //      which segment are we looking at
        int                                             lastPrefixIndex;    //
index of last prefix like segment we got
        int                                             lastSuffixIndex;    //
index of last suffix like segment we got
        int                                             lastNameIndex;
//      index of last non-TAQ segment we got
        int                                             nameSegmentTaqListIndex;
                //      where to put taqs in a name segments taq list
        char                                    *primaryCultureCode =
nameParms->primaryCultureCode;
        char                                    *secondaryCultureCode =
nameParms->secondaryCultureCode;

        //      clear out the TAQ counts for each segment.
        //      This is important because the TAQ segments are not
initalized
        //      if they are not filled in.
        for (i = 0; i < numGnSegments; i++)
                gnSegments[i].numTAQs = 0;

        if      (nameParms->getSeparateGnTaqs() == true)  {
                //      init some variables
                segIndex = 0;
                numTempTAQSegs = 0;

                //      Start out by looking for TAQs at the start of the name
field,
                //      before any name segments.
                //      while there are TAQ values at the start of the gn
                //      get their associated TAQ record and place that in
                //      a temporary list.
                while (segIndex < numGnSegments)    {
                  tempTAQRecordPtr = taqTable-
>getTAQSegment(gnSegments[segIndex].segString,

      primaryCultureCode,

      secondaryCultureCode);
                  if (tempTAQRecordPtr != NULL) {
                        //    make sure we are not past our space for
TAQs in the temp list
                        //    This would happen if a name field started
out with tons of TAQs
                        if (segIndex < NH_MAX_TAQS_PER_SEGMENT)    {
                              tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                              numTempTAQSegs++;
                        }
                        segIndex++;
                  }
                  else
                        break;
            }

            //    as long as we found a non-TAQ segment
            if (segIndex < numGnSegments) {
                  //    fill up the taqList for the first Name Segment
with
                  //    each of the leading TAQs we found.  If we found
no TAQs above,
                  //    numTempTAQSegs will be 0, so we wont even enter
into the loop.
                  //    Also, since we resticted the loop above, we are
guaranteed to
                  //    not exceed our space for TAQs for a single
segment.
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        gnSegments[segIndex].taqList[i].segString =
gnSegments[i].segString;
                        gnSegments[segIndex].taqList[i].taqAction =
tempTAQList[i]->gnAction;
                        gnSegments[segIndex].taqList[i].taqType =
tempTAQList[i]->taqType;
                        gnSegments[segIndex].numTAQs += 1;
                  }

                  //    now move all the segments back starting with
first name segment
                  //    ousting the leading TAQs.  If we found that the
first segment
                  //    was a name segment, we do not need to move
anything.
                  if (segIndex != 0)      {
                        for (i = segIndex; i < numGnSegments;
i++)  {
                              gnSegments[i - segIndex] = gnSegments[i];
                        }
                        //    note that we now have less segments, since
we removed some segments
                        //    that were TAQ values
                        numGnSegments -= segIndex;
                        //    also, set the segIndex to 0, since we are
now back at the begining
                        segIndex = 0;
                  }

                  //    now start looking at the remaining segments
                  //    along the way, we must keep track of
                  //          -     the index of the last Name segment
we found (start out as 0, since we backed it up to 0)
                  //          -     the index of the last "suffix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //          -     the index of the last "prefix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //
                  //    If we get a:
                  //          Name:
                  //                - associate everything between the
lastNameIndex + 1 and the
                  //                      lastSuffixIndex with
gnSegment[lastNameIndex];
                  //                -     associate everything between
the lastPrefixIndex and
                  //                      segIndex - 1 with this name
segment.
                  //                -     move everything back to oust
the TAQ values from the gnSegment array
                  //                -     mark the new lastNameIndex
(lastNameIndex = segIndex;)
                  //                -     adjust numGnSegments for how
many TAQs we ousted
                  //          "Suffix Like"
                  //                lastPrefixIndex = -
1                 //    previous prefix now considered a suffix
                  //                lastSuffixIndex = segIndex
                  //          "Prefix Like"
                  //                lastPrefixIndex =
segIndex
                  //          End of Segments
                  //                - associate everything between the
lastNameIndex + 1 and segIndex
                                          with gnSegment[lastNameIndex];
                  //                -     adjust numGnSegments for how
many TAQs we had at end
                  //
                  //    Note that we do not do any storing of anything
until we either reach the
                  //    end of the sements, or get a non-taq segment.
                  //
                  //    Also, as we read TAQ segments, we store a
pointer to their retrieved
                  //    structure in a list.  We do this because we must
read ahead before
                  //    we can store a TAQs relevant info (type, action)
as being associated
                  //    with a segment, and we do not want to have to
look up the TAQ info twice.
```

```
                  numTempTAQSegs = 0;
                  lastPrefixIndex = -1;
                  lastSuffixIndex = -1;
                  lastNameIndex = segIndex;
                  segIndex++;            //    look at the next segment
                  while (segIndex < numGnSegments)    {
                        tempTAQRecordPtr = taqTable-
>getTAQSegment(gnSegments[segIndex].segString,


            primaryCultureCode,


            secondaryCultureCode);
                        if (tempTAQRecordPtr == NULL) {
                              //    segment is not a TAQ value

                              //    do an initial check to make sure we
actually got one or more TAQs.
                              //    if not, all we really have to do is
just reflect the new value for
                              //    lastNameIndex.
                              if (numTempTAQSegs > 0) {
                                    //    so associate all taqs between
the previous Name segment and
                                    //    the last suffix with the
previous Name Segment.  Since lastSuffixIndex
                                    //    may be -1 (if there we not
suffixes), we may not even enter this for loop.

                                    //    this variable is necessary
because the segment at lastNameIndex
                                    //    might already have TAQs stored
in its taqList (due to prefixes).
                                    //    We must keep track of where
the next available place in that list is.
                                    nameSegmentTaqListIndex =
gnSegments[lastNameIndex].numTAQs;
                                    tempTAQSegIndex = 0;
                                    for (i = lastNameIndex + 1; (i <=
lastSuffixIndex) && (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT);
i++)  {
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].segString = gnSegments[i].segString;
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>gnAction;
                                          gnSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;

                                          tempTAQSegIndex++;
                                          nameSegmentTaqListIndex++;
                                          gnSegments[lastNameIndex].numT
AQs += 1;
                                    }

                                    //    associate everything at or
past the previous prefix(s) with the name
                                    //    segment we just found.  Again,
since there may not have been any
                                    //    prefixes, we might not even
```

```
enter this for loop
                                    if (lastPrefixIndex != -1)    {
                                          for (i = lastPrefixIndex; (i <
segIndex) && (tempTAQSegIndex < NH_MAX_TAQS_PER_SEGMENT); i++)    {
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].segString = gnSegments[i].segString;
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>gnAction;
                                                gnSegments[segIndex].taq
List[i - lastPrefixIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                                tempTAQSegIndex++;
                                                gnSegments[segIndex].num
TAQs += 1;
                                          }
                                    }

                                    //    now move all the segments back
starting with this segment and
                                    //    ending with the last segment.
We move them back to the first
                                    //    segment after the previous
Name segment, which is numTempTAQSegs places
                                    for (i = segIndex; i <
numGnSegments; i++)     {
                                          gnSegments[i - numTempTAQSegs]
= gnSegments[i];
                                    }

                                    //for (i = lastNameIndex + 1; i <
numGnSegments; i++)     {
                                    //    gnSegments[i] = gnSegments[i +
numTempTAQSegs];
                                    //}

                                    numGnSegments -=
numTempTAQSegs;         //    we not have less segments, since we got

//    rid of some TAQs
                                    segIndex -=
numTempTAQSegs;                         //    move our pointer back
too
                                    numTempTAQSegs =
0;                                                //    clear out
the temp segment array
                              }
                              lastNameIndex =
segIndex;                                   //    mark the new
lastNameIndex

                        }
                        else  {
                              if ((tempTAQRecordPtr->taqType == 'P') ||
(tempTAQRecordPtr->taqType == 'T')) {
                                    //    got a prefix or a title
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
                                    //    only set the prefix index if
```

```
we do not have one on record.
                                    //    otherwise, we will only get
the right most prefix in a string
                                    //    of consecutive prefixes.
                                    if (lastPrefixIndex == -1)
                                          lastPrefixIndex = segIndex;
                              }
                              else  {
                                    //    must be a suffix or qualifier
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
                                    lastPrefixIndex = -
1;                //    any previous prefixes now considered a suffix
                                    lastSuffixIndex = segIndex;
                              }
                        }
                        segIndex++;             //    look at next
segment
                  }

                  //    now we are at the end of all segments, so make
sure that any
                  //    TAQs that were trailing get associated with the
last name segment.

                  //    do an initial check to make sure we actually got
one or more TAQs.
                  //    if not, all we really have to do is just reflect
the new value for
                  //    lastNameIndex.
                  if (numTempTAQSegs > 0) {
                        //    associate all the stored taqs with the
last name segment.
                        //    in the loop below:
                        //          i is the index into the gnSegments
list for the TAQ string we are copying
                        //          tempTAQSegIndex is the index into
the tempTAQList for the saved TAQ info
                        //          lastNameIndex is the index into the
gnSegments for the name getting
                        //                the TAQs associated with it.
                        //          gnSegmentTaqListIndex is the index
into the taqList for the name getting
                        //                the TAQs associated with it.
                        //
                        //    We must be careful that we do not
overwrite any TAQs already associated with
                        //    the name (from prefixes).  For this
reason, we use separate indexes for the
                        //    tempTAQList and the gnSegments' taqList.

                        nameSegmentTaqListIndex =
gnSegments[lastNameIndex].numTAQs;
                        tempTAQSegIndex = 0;
                        for (i = lastNameIndex + 1; (i < numGnSegments)
&& (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT); i++)       {
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].segString = gnSegments[i].segString;
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]->gnAction;
```

```
                              gnSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]->taqType;
                              tempTAQSegIndex++;
                              nameSegmentTaqListIndex++;
                              gnSegments[lastNameIndex].numTAQs += 1;
                        }

                        //    now we can just chop off all the TAQ
segments by reducing numGnSegments.
                        numGnSegments -= numTempTAQSegs;
                  }

            }
            else  {
                  //    we did not get any Non-TAQ segments.  Move all
the segments to the TAQ
                  //    list for the first segment, create a single
segment, and set its string
                  //    value to "".
                  gnSegments[0].numTAQs = 0;    //    set this in case
there were no TAQs (empty string)

                              //    In that case, we would not have
cleared it out orignally
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        gnSegments[0].taqList[i].segString =
gnSegments[i].segString;
                        gnSegments[0].taqList[i].taqAction =
tempTAQList[i]->gnAction;
                        gnSegments[0].taqList[i].taqType =
tempTAQList[i]->taqType;
                        gnSegments[0].numTAQs += 1;
                  }
                  numGnSegments = 1;
                  gnSegments[0].segString = "";
                  gnSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
            }
      }

      //    as a last step, we must make sure that the number of
gnSegments is
      //    now no greater than NH_MAX_SEGS_AFTER_TAQ.  We just ignore
any segments
      //    after the max.
      if (numGnSegments > NH_MAX_SEGS_AFTER_TAQ)
            numGnSegments = NH_MAX_SEGS_AFTER_TAQ;


      //    clear out the TAQ counts for each segment.
      //    This is important because the TAQ segments are not
initalized
      //    if they are not filled in.
      for (i = 0; i < numSnSegments; i++)
            snSegments[i].numTAQs = 0;

      //    Now do the SN segments
      if    (nameParms->getSeparateGnTaqs() == true)  {
            //    init some variables
            segIndex = 0;
            numTempTAQSegs = 0;
            //    Start out by looking for TAQs at the start of the name
field,
            //    before any name segments.
            //    while there are TAQ values at the start of the sn
            //    get their associated TAQ record and place that in
            //    a temporary list.
            while (segIndex < numSnSegments)     {
                  tempTAQRecordPtr = taqTable-
>getTAQSegment(snSegments[segIndex].segString,

      primaryCultureCode,

      secondaryCultureCode);
                  if (tempTAQRecordPtr != NULL) {
                        //    make sure we are not past our space for
TAQs in the temp list
                        //    This would happen if a name field started
out with tons of TAQs
                        if (segIndex < NH_MAX_TAQS_PER_SEGMENT)    {
                              tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                              numTempTAQSegs++;
                        }
                        segIndex++;
                  }
                  else
                        break;
            }

            //    as long as we found a non-TAQ segment
            if (segIndex < numSnSegments) {
                  //    fill up the taqList for the first Name Segment
with
                  //    each of the leading TAQs we found.  If we found
no TAQs above,
                  //    numTempTAQSegs will be 0, so we wont even enter
into the loop.
                  //    Also, since we resticted the loop above, we are
guaranteed to
                  //    not exceed our space for TAQs for a single
segment.
                  for (i = 0; i < numTempTAQSegs; i++)       {
                        snSegments[segIndex].taqList[i].segString =
snSegments[i].segString;
                        snSegments[segIndex].taqList[i].taqAction =
tempTAQList[i]->snAction;
                        snSegments[segIndex].taqList[i].taqType =
tempTAQList[i]->taqType;
                        snSegments[segIndex].numTAQs += 1;
                  }

                  //    now move all the segments back starting with
first name segment
                  //    ousting the leading TAQs.  If we found that the
first segment
                  //    was a name segment, we do not need to move
anything.
                  if (segIndex != 0)       {
                        for (i = segIndex; i < numSnSegments;
i++)  {
                              snSegments[i - segIndex] = snSegments[i];
                        }
                        //    note that we now have less segments, since
we removed some segments
                        //    that were TAQ values
                        numSnSegments -= segIndex;

                        //    also, set the segIndex to 0, since we are
now back at the begining
                        segIndex = 0;
                  }

                  //    now start looking at the remaining segments
                  //    along the way, we must keep track of
                  //          -     the index of the last Name segment
we found (start out as 0, since we backed it up to 0)
                  //          -     the index of the last "suffix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //          -     the index of the last "prefix-like"
TAQ we found      (starts out as -1, since all TAQs were tacked onto seg
0)
                  //
                  //    If we get a:
                  //          Name:
                  //                - associate everything between the
lastNameIndex + 1 and the
                  //                      lastSuffixIndex with
snSegment[lastNameIndex];
                  //                -     associate everything between
the lastPrefixIndex and
                  //                      segIndex - 1 with this name
segment.
                  //                -     move everything back to oust
the TAQ values from the snSegment array
                  //                -     mark the new lastNameIndex
(lastNameIndex = segIndex;)
                  //                -     adjust numSnSegments for how
many TAQs we ousted
                  //          "Suffix Like"
                  //                lastPrefixIndex = -
1                 //    previous prefix now considered a suffix
                  //                lastSuffixIndex = segIndex
                  //          "Prefix Like"
                  //                lastPrefixIndex =
segIndex
                  //          End of Segments
                  //                - associate everything between the
lastNameIndex + 1 and segIndex
                  //                      with snSegment[lastNameIndex];
                  //                -     adjust numSnSegments for how
many TAQs we had at end
                  //
                  //    Note that we do not do any storing of anything
until we either reach the
                  //    end of the sements, or get a non-taq segment.
                  //
                  //    Also, as we read TAQ segments, we store a
pointer to their retrieved
                  //    structure in a list.  We do this because we must
```

```
read ahead before
                  //    we can store a TAQs relevant info (type, action)
as being associated
                  //    with a segment, and we do not want to have to
look up the TAQ info twice.

                  numTempTAQSegs = 0;
                  lastPrefixIndex = -1;
                  lastSuffixIndex = -1;
                  lastNameIndex = segIndex;
                  segIndex++;             //    look at the next segment
                  while (segIndex < numSnSegments)    {
                        tempTAQRecordPtr = taqTable-
>getTAQSegment(snSegments[segIndex].segString,

            primaryCultureCode,

            secondaryCultureCode);
                        if (tempTAQRecordPtr == NULL) {
                              //    segment is not a TAQ value

                              //    do an initial check to make sure we
actually got one or more TAQs.
                              //    if not, all we really have to do is
just reflect the new value for
                              //    lastNameIndex.
                              if (numTempTAQSegs > 0) {
                                    //    so associate all taqs between
the previous Name segment and
                                    //    the last suffix with the
previous Name Segment.  Since lastSuffixIndex
                                    //    may be -1 (if there we not
suffixes), we may not even enter this for loop.

                                    //    this variable is necessary
because the segment at lastNameIndex
                                    //    might already have TAQs stored
in its taqList (due to prefixes).
                                    //    We must keep track of where
the next available place in that list is.
                                    nameSegmentTaqListIndex =
snSegments[lastNameIndex].numTAQs;
                                    tempTAQSegIndex = 0;
                                    for (i = lastNameIndex + 1; (i <=
lastSuffixIndex) && (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT);
i++)  {
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].segString = snSegments[i].segString;
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>snAction;
                                          snSegments[lastNameIndex].taqL
ist[nameSegmentTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                          tempTAQSegIndex++;
                                          nameSegmentTaqListIndex++;
                                          snSegments[lastNameIndex].numT
AQs += 1;
```

```
                                    }

                                    //    associate everything at or
past the previous prefix(s) with the name
                                    //    segment we just found.  Again,
since there may not have been any
                                    //    prefixes, we might not even
enter this for loop
                                    if (lastPrefixIndex != -1)    {
                                          for (i = lastPrefixIndex; (i <
segIndex) && (tempTAQSegIndex < NH_MAX_TAQS_PER_SEGMENT); i++)    {
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].segString = snSegments[i].segString;
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].taqAction = tempTAQList[tempTAQSegIndex]-
>snAction;
                                                snSegments[segIndex].taq
List[i - lastPrefixIndex].taqType = tempTAQList[tempTAQSegIndex]-
>taqType;
                                                tempTAQSegIndex++;
                                                snSegments[segIndex].num
TAQs += 1;
                                          }
                                    }

                                    //    now move all the segments back
starting with this segment and
                                    //    ending with the last segment.
We move them back to the first
                                    //    segment after the previous
Name segment, which is numTempTAQSegs places
                                    for (i = segIndex; i <
numSnSegments; i++)    {
                                          snSegments[i - numTempTAQSegs]
= snSegments[i];
                                    }

                                    numSnSegments -=
numTempTAQSegs;        //    we not have less segments, since we got

//    rid of some TAQs
                                    segIndex -=
numTempTAQSegs;                         //    move our pointer back
too
                                    numTempTAQSegs =
0;                                                //    clear out
the temp segment array
                              }
                              lastNameIndex =
segIndex;                           //    mark the new
lastNameIndex

                        }
                        else  {
                              if ((tempTAQRecordPtr->taqType == 'P') ||
(tempTAQRecordPtr->taqType == 'T')) {
                                    //    got a prefix or a title
                                    tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                    numTempTAQSegs++;
```

```
                                     //    only set the prefix index if
we do not have one on record.
                                     //    otherwise, we will only get
the right most prefix in a string
                                     //    of consecutive prefixes.
                                     if (lastPrefixIndex == -1)
                                           lastPrefixIndex = segIndex;
                              }
                              else  {
                                     //    must be a suffix or qualifier
                                     tempTAQList[numTempTAQSegs] =
tempTAQRecordPtr;
                                     numTempTAQSegs++;
                                     lastPrefixIndex = -
1;                //    any previous prefixes now considered a suffix
                                     lastSuffixIndex = segIndex;
                               }
                        }
                        segIndex++;             //    look at next
segment
                  }

                  //    now we are at the end of all segments, so make
sure that any
                  //    TAQs that were trailing get associated with the
last name segment.

                  //    do an initial check to make sure we actually got
one or more TAQs.
                  //    if not, all we really have to do is just reflect
the new value for
                  //    lastNameIndex.
                  if (numTempTAQSegs > 0) {
                        //    associate all the stored taqs with the
last name segment.
                        //    in the loop below:
                        //          i is the index into the snSegments
list for the TAQ string we are copying
                        //          tempTAQSegIndex is the index into
the tempTAQList for the saved TAQ info
                        //          lastNameIndex is the index into the
snSegments for the name getting
                        //                the TAQs associated with it.
                        //          snSegmentTaqListIndex is the index
into the taqList for the name getting
                        //                the TAQs associated with it.
                        //.
                        //    We must be careful that we do not
overwrite any TAQs already associated with
                        //    the name (from prefixes).  For this
reason, we use separate indexes for the
                        //    tempTAQList and the snSegments' taqList.

                        nameSegmentTaqListIndex =
snSegments[lastNameIndex].numTAQs;
                        tempTAQSegIndex = 0;
                        for (i = lastNameIndex + 1; (i < numSnSegments)
&& (nameSegmentTaqListIndex < NH_MAX_TAQS_PER_SEGMENT); i++)      {
                              snSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].segString = snSegments[i].segString;
                              snSegments[lastNameIndex].taqList[nameSegm
```

```
entTaqListIndex].taqAction = tempTAQList[tempTAQSegIndex]->snAction;
                              snSegments[lastNameIndex].taqList[nameSegm
entTaqListIndex].taqType = tempTAQList[tempTAQSegIndex]->taqType;
                              tempTAQSegIndex++;
                              nameSegmentTaqListIndex++;
                              snSegments[lastNameIndex].numTAQs += 1;
                        }

                        //    now we can just chop off all the TAQ
segments by reducing numSnSegments.
                        numSnSegments -= numTempTAQSegs;
                  }

            }
            else  {
                  //    we did not get any Non-TAQ segments.  Move all
the segments to the TAQ
                  //    list for the first segment, create a single
segment, and set its string
                  //    value to "".
                  snSegments[0].numTAQs = 0;     //    set this in case
there were no TAQs (empty string)

                              //    In that case, we would not have
cleared it out orignally
                  for (i = 0; i < numTempTAQSegs; i++)      {
                        snSegments[0].taqList[i].segString =
snSegments[i].segString;
                        snSegments[0].taqList[i].taqAction =
tempTAQList[i]->snAction;
                        snSegments[0].taqList[i].taqType =
tempTAQList[i]->taqType;
                        snSegments[0].numTAQs += 1;
                  }
                  numSnSegments = 1;
                  snSegments[0].segString = "";
                  snSegments[0].status = NH_NAME_FIELD_STATUS_UNKNOWN;
            }
      }

      //    as a last step, we must make sure that the number of
gnSegments is
      //    now no greater than NH_MAX_SEGS_AFTER_TAQ.  We just ignore
any segments
      //    after the max.
      if (numSnSegments > NH_MAX_SEGS_AFTER_TAQ)
            numSnSegments = NH_MAX_SEGS_AFTER_TAQ;

}


//    function to generate index keys for this name.
//    Each key includes a portion for the GN and a portion
//    for the SN.
//    We currently support two key lengths, 32 bits or 64 bits.
//    The GN length does not have to be, the same as the SN length,
//    but GN keys generated must be the same length (similarly for
//    SN).  Thus the full key length could be:
//
//          64:         Both GN and SN are 32 bits
//          96:         Gn is 64, but SN is 32
//          96:         Gn is 32, but SN is 64
//         128:  Both GN and SN are 64 bits
//
//    Keys are generated by name stem segment.  The first key
//    consists of a key for the first GN segment, and a key
//    for the first SN segment.  The second key
//    consists of a key for the second GN segment, and a key
//    for the second SN segment.  When there are a differing number
//    of GN and SN segments, the final segment of the name
//    field with the fewer number of segments is repeated.
//    Thus, the number of keys generated is given by the formula:
//                max(numGnSegs, numSnSegs)
//
//    We do things this way so that a name has the same number of keys
//    for both GN and SN, and in fact we can view the two keys as one
//    contiguous key that can be passed to comparison functions as a
//    single value.
//
//    Note that we are talking about stem segments (TAQ segments have
//    been removed).
//
//    maxKeys specifies how many keys the caller can fit into
//    keyBuff.  It is up to the caller to make sure that they have
allocated
//    enough space in the keyBuff to hold maxKeys.
unsigned    char  NHNameData::genIndexKeys(int maxKeys, NHKeyWidth
gnKeyWidth,

                  NHKeyWidth snKeyWidth, void *keyBuff)
{
      int   numKeysGenerated = 0;
      int gnSegIndex = 0;
      int snSegIndex = 0;
      unsigned    int   *keyPtr = (unsigned     int   *)keyBuff;


      while (numKeysGenerated < maxKeys)  {
            if ((gnSegIndex >= numGnSegments) && (snSegIndex >=
numSnSegments))
                  break;
            else  {
                  numKeysGenerated++;
                  //    make sure that if one segment is now at the end,
                  //    we stay on the last segment
                  if (gnSegIndex == numGnSegments)
                        gnSegIndex--;
                  if (snSegIndex == numSnSegments)
                        snSegIndex--;

                  if (gnKeyWidth == NH_KEY_WIDTH_32)  {
                        //    gn key length is 32
                        *keyPtr =
globalDigraphBitmapArray.get32BitKeyForToken(gnSegments[gnSegIndex].segS
tring);

                        keyPtr++;         //    move the pointer by 4
bytes
                  }
                  else  {
                        //    gn key length is 64
                        globalDigraphBitmapArray.get64BitKeyForToken(gnS
egments[gnSegIndex].segString,

            (bit_64_t *)keyPtr);
                        keyPtr += 2;            //    move the pointer
by 8 bytes
                  }

                  if (snKeyWidth == NH_KEY_WIDTH_32)  {
                        //    gn key length is 32
                        *keyPtr =
globalDigraphBitmapArray.get32BitKeyForToken(snSegments[snSegIndex].segS
tring);
                        keyPtr++;         //    move the pointer by 4
bytes
                  }
                  else  {
                        //    gn key length is 64
                        globalDigraphBitmapArray.get64BitKeyForToken(snS
egments[snSegIndex].segString,

            (bit_64_t *)keyPtr);
                        keyPtr += 2;            //    move the pointer
by 8 bytes
                  }

                  //    advance the segment indexes
                  snSegIndex++;
                  gnSegIndex++;
            }
      }
      return numKeysGenerated;
}
```

```
//    File:  NHEvalNameData.cpp
//
//    Description:
//
//          Implementation to the NHEvalNameData class.
//
//
//    History:
//
//          5/14/97          EFB          Created
//          9/1/97           EFB          Lots of changes to support
retaining segment scores in
//                                                          best mode so
that sorting can be more detailed and accurate
//          10/31/97    EFB          Made several member functions
protected, and made performComp()
//                                                          a friend of
NHQueryNameData.  Also changed performComp to
//                                                          NOT delete
objects that are not passed on to the resultslist,
//                                                          to
accomodate the new method of deleting NHEvalNameData objects.
//          11/03/97    EFB          Added a new function,
calcNameScore() and made it virtual.
//                                                          removed
virtual from performComp.  The perform comp method
//                                                          was too
complicated to be subclassed.  We really only want
//                                                          callers to
be able to affect the name score and the determination
//                                                          of
HIT/NO_HIT.  These are now the only virtual functions.  Both
//                                                          are now
inline in the header file so the caller knows exactly
//                                                          what is
happening in these functions if they decide to subclass
//                                                          and
override. OOPS, I forgot compareScore(), which is also
//                                                          virtual - we
want them to be able to change how hits are sorted.
//
//          3/02/98     EFB               Made lots of changes necessary
when I moved a bunch of
//                                                          parameters
(the ones associated with parsing the name)
//                                                          from the
NHCompParms class into a new class called NHNameParms.
//                                                          and renamed
the NHCompParms class to NHCompParms.
//          3/20/98     EFB          Changed names to NH from SN


#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include    "NHEvalNameData.hpp"
#include    "NHQueryNameData.hpp"
#include    "NH_util.hpp"
#include    "NH_queens_arrays.hpp"
```

```
#include    "NHVariantTable.hpp"
#include    "NHResultsList.hpp"
#include    "NHTAQTable.hpp"
#include    "NHNameParms.hpp"


//    private, non-member function prototype
static double     NH_digraph_score(char *qSeg, int qSegLen,

char *evalSeg, int evalSegLen,
                         bool useLeftDigraphBias);

static      double      NH_best_score(int numQSegs, int numEvalSegs,
NHSegScoreMode scoreMode,
                                                                  double

scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ]);

void  NH_best_score_for_highest_mode(int xDim, int yDim, double
highestScore,
                                                                  double

*bestSegScores,
                                                                  double

scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ]);

static      double NH_calc_score(   SegList qSegs, int numQSegs,
                                                                  SegLis

t evalSegs, int numEvalSegs,
                                                                  SegLis

tVariants querySegmentVariants,
                                                                  char

                                    *primaryCulture,
                                                                  char

                                    *secondaryCulture,
                                                                  NHComp

Parms *compParms,
                                                                  NHName

Parms *nameParms,
                                                                  NHName

Fields nameField,
                                                                  char

*origQNameField,
                                                                  char

*origEvalNameField,
                                                                  int

*numSegsScored,
                                                                  double

*bestSegScores);

static      void  NH_apply_TAQs_to_score(double *diScore, Segment *qSeg,
Segment *evalSeg,

                        double absDelTAQFactor,

                        double absDisTAQFactor,

                        double delTAQFactor,

                        double disTAQFactor);

static      bool NH_check_compressed_name(char *qSegString, char
*evalSegString,
                                        char *compressChars Part1,
                                        char  *compressCharsPart2);


NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *aGn, char
*aSn) :
                                             NHNameData(nParms, aGn,
aSn)
{
      resetScores();
}


NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *aGn, char
*aSn, char *aMn) :
                                             NHNameData(nParms, aGn,
aSn, aMn)
{
      resetScores();
}


NHEvalNameData::NHEvalNameData(NHNameParms *nParms, char *name,
NHNameFormat nameFormat)      :
                                             NHNameData(nParms, name,
nameFormat)
{
      resetScores();
}


//    constuct an object from an archived representation in
//    a stream.
//
//    The archive is in the following order
//
//    gnLen
//    snLen
//    nameStorage
NHEvalNameData::NHEvalNameData(NHNameParms *nParms, istream &inStream) :
                                             NHNameData(nParms,
inStream)
{
      //    read the gn, sn and name scores
      if (inStream)
            inStream.read((char *)&gnScore, sizeof(gnScore));
      if (inStream)
            inStream.read((char *)&snScore, sizeof(snScore));
      if (inStream)
            inStream.read((char *)&nameScore, sizeof(nameScore));

      //    seg differentials
      if (inStream)
            inStream.read((char *)&gnSegDifferential,
sizeof(gnSegDifferential));
      if (inStream)
            inStream.read((char *)&snSegDifferential,
sizeof(snSegDifferential));
```

```
      //    read the number of gn segs scored, and however many scores
we need     inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
      if (inStream)
            inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
      if (inStream)     {
            if (numGnSegsScored > 0)      {
                  inStream.read((char *)gnSegScores, numGnSegsScored *
sizeof(double));
            }
      }

      //    read the number of sn segs scored, and however many scores
we need
      if (inStream)
            inStream.read((char *)&numSnSegsScored,
sizeof(numSnSegsScored));
      if (inStream)     {
            if (numSnSegsScored > 0)      {
                  inStream.read((char *)snSegScores, numSnSegsScored *
sizeof(double));
            }
      }
}


NHEvalNameData::~NHEvalNameData()
{
}


bool  NHEvalNameData::archiveData(ostream &outStream)
{
      bool  rc = true;


      rc = NHNameData::archiveData(outStream);
      if (rc)     {
            //    read the gn, sn and name scores
            outStream.write((char *)&gnScore, sizeof(gnScore));
            outStream.write((char *)&snScore, sizeof(snScore));
            outStream.write((char *)&nameScore, sizeof(nameScore));

            //    seg differentials
            outStream.write((char *)&gnSegDifferential,
sizeof(gnSegDifferential));
            outStream.write((char *)&snSegDifferential,
sizeof(snSegDifferential));

            //    read the number of gn segs scored, and however many
scores we need    inStream.read((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
            outStream.write((char *)&numGnSegsScored,
sizeof(numGnSegsScored));
            if (numGnSegsScored > 0)      {
                  outStream.write((char *)gnSegScores, numGnSegsScored *
sizeof(double));
            }
            //    read the number of sn segs scored, and however many
scores we need
            outStream.write((char *)&numSnSegsScored,
sizeof(numSnSegsScored));
            if (numSnSegsScored > 0)       {
                  outStream.write((char *)snSegScores, numSnSegsScored *
sizeof(double));
            }
      }

      return rc;
}


//    note that this function is a friend of NHQueryNameData, which is
//    why we are able to access private member functions of that class.
void  inline NHEvalNameData::calcComponentScores(NHQueryNameData
*queryName)
{
      char                              *primaryCulture = nameParms-
>primaryCultureCode;
      char                              *secondaryCulture = nameParms-
>secondaryCultureCode;

      //    do the digraph compare and set the scores
      gnScore = NH_calc_score(queryName->gnSegments, queryName-
>numGnSegments,
                                                                  gnSegm
ents, numGnSegments,
                                                                  queryN
ame->gnSegmentVariants,
                                                                  primar
yCulture, secondaryCulture,
                                                                  compPa
rms,
                                                                  namePa
rms,
                                                                  NH_FIR
ST_NAME,
                                                                  queryN
ame->gn, gn,
                                                                  &numGn
SegsScored,
                                                                  gnSegS
cores);
      snScore = NH_calc_score(queryName->snSegments, queryName-
>numSnSegments,
                                                                  snSegm
ents, numSnSegments,
                                                                  queryN
ame->snSegmentVariants,
                                                                  primar
yCulture, secondaryCulture,
                                                                  compPa
rms,
                                                                  namePa
rms,
                                                                  NH_LAS
T_NAME,
                                                                  queryN
ame->sn, sn,
                                                                  &numSn
SegsScored,
                                                                  snSegS
cores);
}

//    note that this function is a friend of NHQueryNameData, which is
//    why we are able to access private member functions of that class.
NHReturnCode      NHEvalNameData::performComp(NHQueryNameData
*queryName,

                                                            NHCompParms
*someCompParms)
{
      NHReturnCode            compResult;
      NHResultsList           *resultList;


      //    save the compParms so that they can be easily referenced
      //    throughout the comparison process.
      compParms = someCompParms;

      calcComponentScores(queryName);

      //    call a method to calculate the name score.
      calcNameScore();

      //    store the segments differentials, in case we get a tie
score.
      gnSegDifferential = abs(numGnSegments - queryName-
>getNumGnSegments());
      snSegDifferential = abs(numSnSegments - queryName-
>getNumSnSegments());

      //    Now call the getCompResult() function to get the return
value
      //    (i.e. was it a match?)
      compResult = getCompResult();

      //    now see if we are working with a results list
      resultList = queryName->getResultsList();
      if (resultList != NULL) {
            //    we are using a result list.  If this is a hit, add it
            //    to the result list.
            //    Otherwise, delete it
            if (compResult == NH_MATCH)   {
                  NHReturnCode            tempInsertResult;

                  //    make sure the insert works.  If so, don't mess
with
                  //    the compResult, so the comparison will be
returned
                  //    as a hit.  If there was an error, delete this
object,
                  //    and save the error code so it can be returned.
                  tempInsertResult = resultList->addHit(this);
                  if (tempInsertResult != NH_SUCCESS) {
                        compResult = tempInsertResult;
                  }
            }
```

```
        }
        return compResult;
}


//      used only when the segment mode is set to HIGHEST.
//      It compares the segment scores the were retained when
//      the name was compared to the query name.
//      We are comparing the segment scores for two (pre-scored)
//      eval names.  The comparison should find which name has
//      the "best" set of segment scores, where best is defined
//      as "the one with the highest best score".  If the best
//      score results in a tie, we move on to the second best score,
//      and so on until we find a difference, or there are no more
//      segments to compare.  Each name has variables numGnSegsScored
//      and numSnSegsScored, that tell how many segments were scored
//      in the name.  We do up to N comparisons, where N is the larger
//      of the number of segments scored in each name.  Where one name
//      has less segments scored than the other, a default value of
//      NH_DEFAULT_MISSING_SEGMENT_SCORE is assigned.  This is so that
//      a scored segment has to beat some threshold to be considered
//      better than nothing at all.
//
double          NHEvalNameData::compareSegmentScores(NHEvalNameData
*scoredName, NHNameFields nameField)
{
        double          scoreDiff;
        int                     maxComparisons;
        double          *thisEvalScores;
        double          *compEvalScores;
        int                     numSegsScoredForThisEval;
        int                     numSegsScoredForCompEval;


        if (nameField == NH_LAST_NAME)          {
                thisEvalScores = snSegScores;
                compEvalScores = scoredName->snSegScores;
                numSegsScoredForThisEval = numSnSegsScored;
                numSegsScoredForCompEval = scoredName->numSnSegsScored;
        }
        else    {
                thisEvalScores = gnSegScores;
                compEvalScores = scoredName->gnSegScores;
                numSegsScoredForThisEval = numGnSegsScored;
                numSegsScoredForCompEval = scoredName->numGnSegsScored;
        }
        maxComparisons = numSegsScoredForThisEval >
numSegsScoredForCompEval ? numSegsScoredForThisEval :
numSegsScoredForCompEval;

        for (int i = 0; i < maxComparisons; i++)   {
                if (i >= numSegsScoredForThisEval)
                        thisEvalScores[i] = NH_DEFAULT_MISSING_SEGMENT_SCORE;
                else            //      we can do an else because only one segment
can be missing, not both
                        if (i >= numSegsScoredForCompEval)
                                compEvalScores[i] =
NH_DEFAULT_MISSING_SEGMENT_SCORE;

                scoreDiff = compEvalScores[i] - thisEvalScores[i];
                if (scoreDiff != 0)
                break;
        }

        return scoreDiff;
}

/*********************************************************************
****/

/* NH_calc_score
   Performs a string comparison on two name fields.
        Returns a value between 0.00 and
    1.00, with 1.00 being an exact-fit

*/
double NH_calc_score(   SegList qSegs, int numQSegs,                    SegLis

t evalSegs, int numEvalSegs,                                            SegLis

tVariants querySegmentVariants,                                         char
                                        *primaryCulture,
                                                                        char
                                        *secondaryCulture,
                                                                        NHComp
 Parms *compParms,                                                      NHName

 Parms *nameParms,                                                      NHName

 Fields nameField,                                                      char

 *origQNameField,                                                       char

 *origEvalNameField,                                                    int

 *numSegsScored,                                                        double

 *bestSegScores)
 {
        NHAnchorSegMode         anchorSeg;
        NHSegScoreMode          scoreMode;
                                                        oopsFactor;
        double                                          absDelTAQFactor;
        double                                          absDisTAQFactor;
        double                                          delTAQFactor;
        double                                          disTAQFactor;
        double                                          matchInit;
        bool                                            initScore;
        double                                          initialOnInitialMatchSco
        double
 re;                                                    checkVariant;
        bool                                                    variantScore;
        //      double                                  leftDigraphBias;
        bool                                            anchorFactor;
        double                                          nameUnknownScore;
        double                                          noNameScore;
        double                                      scoresTable[NH_MAX_SEGS_AFTER_
   double
 TAQ][NH_MAX_SEGS_AFTER_TAQ];  //     scores for segment pairs
                                                                qIndex;
        int
        //    temp index for query segments
```

```
     int                                                     evalIndex;  //
temp index for eval segments
     int                                                     qSegLen;
//    hold string length of query segment
     int                                                     evalSegLen; //
hold string length of eval segment
  double                                               diScore =
0.0;        //    temp score for single pair comparison
     double                                                  hiScore =
0.0;        //    temp score to hold best score as we iterate,

                                                       //    which lets us avoid
best_score in mode=BEST
     bool                                                    areVariants;
//    temp flag to hold if the pair are variants
  double                                                     returnValue = 0.0;
     NHVariantTable                                    *variantTable;
     double                                                       varScore;
     NHVarId                                                      evalSegVarId
;
     bool                                                    scoreTaqs;
     double                                                  compressedNameScore;
     bool                                                    checkCompressedName;


     //    set some paramters based on the name field
  if (nameField == NH_LAST_NAME) {
    anchorSeg = compParms->getSnAnchorSegmentMode();
    scoreMode = compParms->getSnSegmentScoreMode();
    oopsFactor = compParms->getSnOOPSFactor();
    matchInit = compParms->getMatchSnIntial();
    initScore = compParms->getSnInitialScore();
            initialOnInitialMatchScore = compParms-
>getSnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseSnVariants();
            anchorFactor = compParms->getSnAnchorFactor();
            leftDigraphBias = compParms->getUseSnLeftBias();
            nameUnknownScore = compParms->getLNUScore();
            noNameScore = compParms->getNLNScore();
            scoreTaqs = compParms->getScoreSnTAQs();
            absDelTAQFactor = compParms->getAbsDelSnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisSnTAQFactor();
            delTAQFactor = compParms->getDelSnTAQFactor();
            disTAQFactor = compParms->getDisSnTAQFactor();
            compressedNameScore = compParms->getSnCompressedNameScore();
            checkCompressedName = compParms->getCheckSnCompressedName();
            variantTable = nameParms->snVariantTable;
     }
  else {
    anchorSeg = compParms->getGnAnchorSegmentMode();
    scoreMode = compParms->getGnSegmentScoreMode();
    oopsFactor = compParms->getGnOOPSFactor();
    matchInit = compParms->getMatchGnIntial();
    initScore = compParms->getGnInitialScore();
            initialOnInitialMatchScore = compParms-
>getGnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseGnVariants();
            anchorFactor = compParms->getGnAnchorFactor();
            leftDigraphBias = compParms->getUseGnLeftBias();
            nameUnknownScore = compParms->getFNUScore();
            noNameScore = compParms->getNFNScore();
```

```
      int                                                   evalIndex;  //
temp index for eval segments
      int                                                   qSegLen;
//    hold string length of query segment
      int                                                   evalSegLen; //
hold string length of eval segment
  double                                              diScore =
0.0;        //    temp score for single pair comparison
      double                                                hiScore =
0.0;        //    temp score to hold best score as we iterate,

                                                      //    which lets us avoid
best_score in mode=BEST
      bool                                                  areVariants;
//    temp flag to hold if the pair are variants
  double                                                    returnValue = 0.0;
      NHVariantTable                                  *variantTable;
      double                                                      varScore;
      NHVarId                                                     evalSegVarId
;
      bool                                                  scoreTaqs;
      double                                                compressedNameScore;
      bool                                                  checkCompressedName;


      //    set some paramters based on the name field
  if (nameField == NH_LAST_NAME) {
    anchorSeg = compParms->getSnAnchorSegmentMode();
    scoreMode = compParms->getSnSegmentScoreMode();
    oopsFactor = compParms->getSnOOPSFactor();
    matchInit = compParms->getMatchSnIntial();
    initScore = compParms->getSnInitialScore();
            initialOnInitialMatchScore = compParms-
>getSnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseSnVariants();
            anchorFactor = compParms->getSnAnchorFactor();
            leftDigraphBias = compParms->getUseSnLeftBias();
            nameUnknownScore = compParms->getLNUScore();
            noNameScore = compParms->getNLNScore();
            scoreTaqs = compParms->getScoreSnTAQs();
            absDelTAQFactor = compParms->getAbsDelSnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisSnTAQFactor();
            delTAQFactor = compParms->getDelSnTAQFactor();
            disTAQFactor = compParms->getDisSnTAQFactor();
            compressedNameScore = compParms->getSnCompressedNameScore();
            checkCompressedName = compParms->getCheckSnCompressedName();
            variantTable = nameParms->snVariantTable;
      }
  else {
    anchorSeg = compParms->getGnAnchorSegmentMode();
    scoreMode = compParms->getGnSegmentScoreMode();
    oopsFactor = compParms->getGnOOPSFactor();
    matchInit = compParms->getMatchGnIntial();
    initScore = compParms->getGnInitialScore();
            initialOnInitialMatchScore = compParms-
>getGnInitialOnInitialMatchScore();
    checkVariant = compParms->getUseGnVariants();
            anchorFactor = compParms->getGnAnchorFactor();
            leftDigraphBias = compParms->getUseGnLeftBias();
            nameUnknownScore = compParms->getFNUScore();
            noNameScore = compParms->getNFNScore();
      int                                                   evalIndex;  //
temp index for eval segments
      int                                                   qSegLen;
//    hold string length of query segment
      int                                                   evalSegLen; //
hold string length of eval segment
   double                                       diScore =
0.0;        //    temp score for single pair comparison
      double                                            hiScore =
0.0;        //    temp score to hold best score as we iterate,

                                                //    which lets us avoid
best_score in mode=BEST
      bool                                              areVariants;
//    temp flag to hold if the pair are variants
   double                                               returnValue = 0.0;
      NHVariantTable                            *variantTable;
      double                                                     varScore;
      NHVarId                                                    evalSegVarId
;
      bool                                              scoreTaqs;
      double                                            compressedNameScore;
      bool                                              checkCompressedName;


      //    set some paramters based on the name field
   if (nameField == NH_LAST_NAME) {
     anchorSeg = compParms->getSnAnchorSegmentMode();
     scoreMode = compParms->getSnSegmentScoreMode();
     oopsFactor = compParms->getSnOOPSFactor();
     matchInit = compParms->getMatchSnIntial();
     initScore = compParms->getSnInitialScore();
            initialOnInitialMatchScore = compParms-
>getSnInitialOnInitialMatchScore();
     checkVariant = compParms->getUseSnVariants();
            anchorFactor = compParms->getSnAnchorFactor();
            leftDigraphBias = compParms->getUseSnLeftBias();
            nameUnknownScore = compParms->getLNUScore();
            noNameScore = compParms->getNLNScore();
            scoreTaqs = compParms->getScoreSnTAQs();
            absDelTAQFactor = compParms->getAbsDelSnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisSnTAQFactor();
            delTAQFactor = compParms->getDelSnTAQFactor();
            disTAQFactor = compParms->getDisSnTAQFactor();
            compressedNameScore = compParms->getSnCompressedNameScore();
            checkCompressedName = compParms->getCheckSnCompressedName();
            variantTable = nameParms->snVariantTable;
      }
   else {
     anchorSeg = compParms->getGnAnchorSegmentMode();
     scoreMode = compParms->getGnSegmentScoreMode();
     oopsFactor = compParms->getGnOOPSFactor();
     matchInit = compParms->getMatchGnIntial();
     initScore = compParms->getGnInitialScore();
            initialOnInitialMatchScore = compParms-
>getGnInitialOnInitialMatchScore();
     checkVariant = compParms->getUseGnVariants();
            anchorFactor = compParms->getGnAnchorFactor();
            leftDigraphBias = compParms->getUseGnLeftBias();
            nameUnknownScore = compParms->getFNUScore();
            noNameScore = compParms->getNFNScore();
            scoreTaqs = compParms->getScoreGnTAQs();
            absDelTAQFactor = compParms->getAbsDelGnTAQFactor();
            absDisTAQFactor = compParms->getAbsDisGnTAQFactor();
            delTAQFactor = compParms->getDelGnTAQFactor();
            disTAQFactor = compParms->getDisGnTAQFactor();
            compressedNameScore = compParms->getGnCompressedNameScore();
            checkCompressedName = compParms->getCheckGnCompressedName();
            variantTable = nameParms->gnVariantTable;
  }

      //    clear out the scores table
  for (qIndex = 0; qIndex < NH_MAX_SEGS_AFTER_TAQ; ++qIndex)
    for (evalIndex = 0; evalIndex < NH_MAX_SEGS_AFTER_TAQ; ++evalIndex)
      scoresTable[qIndex][evalIndex] = 0.0;

      //    now go through each possible combination of segment pairs
      //    (created by matching a query segment against an eval
segment).
      //    Store the scores in the scoresTable.
  for (qIndex = 0; qIndex < numQSegs; ++qIndex) {

    qSegLen = strlen(qSegs[qIndex].segString);

    for (evalIndex = 0; evalIndex < numEvalSegs; ++evalIndex) {

      evalSegLen = strlen(evalSegs[evalIndex].segString);

                  //    first check for either the query or eval segment
being
                  //    blank.
                  if ((qSegLen == 0) || (evalSegLen == 0))  {
                        //    We make a distinction between "unknown"
                        //    and "none".  The table below shows the
scores
                        //    we assign for the various combinations of
Known - K,
                        //    Unknown - U, and None -N.
                        //
                        //          |     K
      |     U
       |    N
                        //    ------------------------------------------
-------------------------
                        //    K     |     N/A
|     unknownScore                              |    NoneScore
                        //    ------------------------------------------
-------------------------
                        //    U     |     unknownScore|     (unknownScor
e + 1) / 2 |      (unknownScore + 1) / 2
                        //    ------------------------------------------
-------------------------
                        //    N     |     NoneScore         |     (unkno
wnScore + 1) / 2 |      (NoneScore + 1) / 2
                        //    ------------------------------------------
-------------------------


                        if (qSegs[qIndex].status ==
NH_NAME_FIELD_STATUS_KNOWN)   {
                              //    we should not need to check for both
being known
```

```
                              if (evalSegs[evalIndex].status ==
NH_NAME_FIELD_STATUS_UNKNOWN)
                                    diScore = nameUnknownScore;
                              else  //    must be
NH_NAME_FIELD_STATUS_NON_EXISTANT
                                    diScore = noNameScore;
                        }
                        else if (qSegs[qIndex].status ==
NH_NAME_FIELD_STATUS_UNKNOWN) {
                              if (evalSegs[evalIndex].status ==
NH_NAME_FIELD_STATUS_KNOWN)
                                    diScore = nameUnknownScore;
                              else if (evalSegs[evalIndex].status ==
NH_NAME_FIELD_STATUS_UNKNOWN)
                                    diScore = (nameUnknownScore + 1.0) /
2.0;
                              else  //    must be
NH_NAME_FIELD_STATUS_NON_EXISTANT,  same score as
                                                //    above, but we
repeat it in case we cange behavior later
                                    diScore = (nameUnknownScore + 1.0) /
2.0;
                        }
                        else  {     //    query must be
NH_NAME_FIELD_STATUS_NON_EXISTANT)
                              if (evalSegs[evalIndex].status ==
NH_NAME_FIELD_STATUS_KNOWN)
                                    diScore = noNameScore;
                              else if (evalSegs[evalIndex].status ==
NH_NAME_FIELD_STATUS_UNKNOWN)
                                    diScore = (nameUnknownScore + 1.0) /
2.0;
                              else  //    must be
NH_NAME_FIELD_STATUS_NON_EXISTANT,  same score as
                                                //    above, but we
repeat it in case we cange behavior later
                                    diScore = (noNameScore + 1.0) / 2.0;
                        }
                  }
                  else  {
                        //    check the variants if
                        //          -     we are supposed to
                        //          -     we have a list of variants to
check
                        //          -     there is a variant for this
segment of the query
                        //          Note we must check the secondary
variants if the
                        //          primary check does not find a
variant.
                        areVariants = false;
                        if (checkVariant && (querySegmentVariants !=
NULL) &&
                                    (querySegmentVariants[qIndex] !=
NULL))      {
                              //    so see if the eval name segment has
any variants in the variant table
                              evalSegVarId = variantTable-
>getVariantIdForName(evalSegs[evalIndex].segString);
                              if (evalSegVarId !=
NH_VAR_NOT_FOUND) {
```

```
                                    //    yes, it did have some
variants, so see if there is an intersection
                                    varScore =
querySegmentVariants[qIndex]-
>getVariantScoreForIdAndCulture(evalSegVarId, primaryCulture);
                                    if (varScore !=
NH_VARIANTS_NOT_RELATED)      {
                                          areVariants = true;
                                          diScore = varScore;
                                    }
                                    else  {
                                          //    variants were not
related, so check for the secondary
                                          //    variant source
                                          //    Put a check in here to
see if the primary culture
                                          //    code was
NH_CULTURE_CODE_GENERIC.  If so, we can skip this check
                                          //    since the secondary code
is always generic
                                          if (strcmp(primaryCulture,
NH_CULTURE_CODE_GENERIC))      {
                                                varScore =
querySegmentVariants[qIndex]-
>getVariantScoreForIdAndCulture(evalSegVarId, secondaryCulture);
                                                if (varScore !=
NH_VARIANTS_NOT_RELATED)      {
                                                      areVariants =
true;
                                                      diScore =
varScore;
                                                }
                                          }
                                    }
                              }
                        }

                        //    now, if we did not find variants above,
check for intials
                        // do we have an initial and are we supposed to
check them?
                        if (areVariants == false)     {
                              if (matchInit && (qSegLen == 1 ||
evalSegLen == 1)) {
                                    //    does the first char match ?
                                    if (qSegs[qIndex].segString[0] ==
evalSegs[evalIndex].segString[0])    {
                                          //    if the second char
matches, we have an initial on inital match,
                                          //    since we know the length
of atleast one of them is 1.
                                          if (qSegs[qIndex].segString[1]
== evalSegs[evalIndex].segString[1])
                                                diScore =
initialOnInitialMatchScore;
                                          else        //    initial
match, but one was more than a single character
                                                diScore =
initScore;       //    so assign initScore
                                    }
                                    else
```

```
                                          diScore =
0.0;              //    no match at all, since first char was off
                              }
                              else {                 //    else not initials
or we shouldn't check them
                                          //    when here, we do not
have unknowns, variants, or initials,
                                          //    so do a digraph
comparison.
                                          diScore =
NH_digraph_score(qSegs[qIndex].segString, qSegLen,

evalSegs[evalIndex].segString, evalSegLen,

                                           leftDigraphBias);
                              }
                        }           //    end, if (areVariants == false)
                  }           //    end, else, both segs are known
(neither name is blank)

                  /*
                        Here we need to handle the oops and anchor
segment parameters.
                        oops specifies a factor to multiply by the score
when the segments
                        are not in the same position.
                        AnchorSeg, AnchorFactor specify a factor to
multiply matches that
                        are in the same segment position, but are in a
segment other than
                        the stated AnchorSeg.  Note that AnchorSeg does
not get applied in
                        average mode, because otherwise a 2 segment name
that was
                        an exact match would get less than 1.0, since
the segment that
                        was not in the anchor segment would be
penalized.  Anchor Factor
                        is meant more to provide a penalty when a
(relatively)
                        unimportant segment is used as the sole
contributor to
                        the score.

                        Note that only one of the factors may be
applied, since oops only
                        gets applied to segments that are out of place,
and anchorFactor
                        only gets applied to matches that are in place.

                        AnchorSeg is also used to determine segment
alignment. anchorSeg
                        value 1 indicates segments should be lined up on
the left, while
                        value 2 indicates they should be lined up on the
right.  A value
                        of 0 indicates they should be lined up on the
left (this is the
                        default.
                  */
```

```
                    switch (anchorSeg) {
                          case 0                                        //    no
:
anchor segment designation
                                if (qIndex != evalIndex)        //    out of
place, so apply oops
                                      diScore *= oopsFactor;
                                break;
                          case 1                                        //    first
:
segment is most important
                                if (qIndex !=
evalIndex)        //    out of place, so apply oops
                                      diScore *= oopsFactor;
                                else
                                      if ((qIndex != 0) && (scoreMode !=
NH_SEGMODE_AVG))        //    if not the first segment (anchor seg)
                                            diScore *=
anchorFactor;           //    apply the anchorFactor, so long as the
                                break;
                                      //    scoreMode is not
NH_SEGMODE_AVG
                          case 2 : /* If not last-to-last match... */
                                if ((qIndex == numQSegs - 1) && (evalIndex
== numEvalSegs - 1))
                                      ; // no modification, since both are
end segments
                                else  {
                                      //    see if they are in the same
position, counting back from the end
                                      if ((numQSegs - qIndex) ==
(numEvalSegs - evalIndex))
                                            if (scoreMode !=
NH_SEGMODE_AVG)         //    skip anchor factor in average seg mode
                                                  diScore *=
anchorFactor;           //    apply the anchorFactor
                                      else
                                            diScore *= oopsFactor;
                                }
                                break;
                    }

                    //    Now we need to apply the TAQ values to the
score,
                    //    but only if they wanted to, and we have a score
                    //.   greater than 0 (otherwise, factors have no
effect).
                    if ((scoreTaqs) && (diScore > 0.0))
                          NH_apply_TAQs_to_score(&diScore, &qSegs[qIndex],
&evalSegs[evalIndex],

                          absDelTAQFactor, absDisTAQFactor,

                          delTAQFactor, disTAQFactor);

                    if (numQSegs > numEvalSegs)          //    always store
smaller dimension as rows
                          scoresTable[evalIndex][qIndex] = diScore;
                    else
                          scoresTable[qIndex][evalIndex] = diScore;
```

```
                  hiScore = hiScore > diScore ? hiScore : diScore;

            } // for evalIndex

      } // for qIndex


      //    now figure out a composite score from all the best scores
      //    Note that for Best score, we must set the number of segments
      //    that were scored, and fill an array containing those scores
-
      //    these will be used later to sort hits).
      //    The exception to this is when either the query or the
      //    eval name field has just 1 segment, in which case we only
      //    score one segment, which becomes the score (in all modes).
      if ((numEvalSegs == 1) || (numQSegs == 1))      {
            if (scoreMode == NH_SEGMODE_HIGHEST)      {
                  *numSegsScored = 1;           //    note that we only
scored 1 segment
                  bestSegScores[0] = hiScore;         //    save the
singly scored segment
            }
            returnValue = hiScore;
      }
      else  {
            //    both have more than 1 segment
            if (numQSegs > numEvalSegs)   {           //    always call
functions with smaller dimension as rows
                  if (scoreMode == NH_SEGMODE_HIGHEST)      {
                        NH_best_score_for_highest_mode(numEvalSegs,
numQSegs, hiScore, bestSegScores, scoresTable);
                        *numSegsScored = numEvalSegs;       //    note
that we only scored numEvalSegs segments
                        returnValue = hiScore;
                  }
                  else
                        returnValue = NH_best_score(numEvalSegs,
numQSegs, scoreMode, scoresTable);
            }
            else  {
                  if (scoreMode == NH_SEGMODE_HIGHEST)      {
                        NH_best_score_for_highest_mode(numQSegs,
numEvalSegs, hiScore, bestSegScores, scoresTable);
                        *numSegsScored = numQSegs;          //.   note
that we only scored numQSegs segments
                        returnValue = hiScore;
                  }
                  else
                        returnValue = NH_best_score(numQSegs,
numEvalSegs, scoreMode, scoresTable);
            }
  }


      //    here we need to see if we are suppoed to check compressed
names.
      //    if so, we have to call the NH_check_compressed_name()
function.
      //    If that function returns true, we pick the higher of the
      //    compressedScore (which is a parameter) and the current
returnValue.
```

```
      if (checkCompressedName &&
                       NH_check_compressed_name(origQNameField,
origEvalNameField,

                             nameParms->getSegmentBreakChars(),

                             nameParms->getNoiseChars()))
                  returnValue = returnValue > compressedNameScore ?
returnValue : compressedNameScore;

  return returnValue;

} /* NH_calc_score */


/* NH_check_compressed_name

      Compresses both names passed in, and sees if they are exact
matches.

      The compression is implemented by skipping characters specified in
      compressChars.
*/
bool NH_check_compressed_name(char *qSegString, char *evalSegString,
char *compressCharsPart1,

                  char  *compressCharsPart2)
{
      char  compressedQuerySeg[NH_MAX_SEG_LENGTH + 1];
      char  compressedEvalSeg[NH_MAX_SEG_LENGTH + 1];
      char  compressChars[200 + 1];
  char *p;
  char  *q;


      //    first, combine the compressCharsPart1 and compressCharsPart1
strings
      strcpy(compressChars, compressCharsPart1);
      strcat(compressChars, compressCharsPart2);

      //    compress the query segment
  for (p = qSegString, q = compressedQuerySeg; *p != EOS; p++)
    if (strchr(compressChars, *p) == NULL)
      *q++ = *p;
  *q = EOS;

      //    compress the query segment
  for (p = evalSegString, q = compressedEvalSeg; *p != EOS; p++)
    if (strchr(compressChars, *p) == NULL)
      *q++ = *p;
  *q = EOS;

      //    at this point, we are not necessarily upper cased, so ignore
case
      //    during the string copy
      return !strcasecmp(compressedQuerySeg, compressedEvalSeg);

} /* NH_check_compressed_name */
```

```
/* NH_best_score
            From a matrix of scores compute the highest possible
combination
            of scores.  During the evaluation of the matrix, a given row
or
            column must provide one and only one score.

            We use a mode to determine how we calculate a score.  The
mode
            can be either NH_SEGMODE_AVG or NH_SEGMODE_LOWEST.  Both
modes
            start out by selecting the combination of values (with no
row or
            column being used more than once) that gives the highest
sum.  Then,
            for mode = NH_SEGMODE_AVG, the final score is the average of
all
            these scores.  For NH_SEGMODE_LOWEST, it is the worst of
these scores.

            If the matrix is non-square (x <> y), our final score
calculation
            only includes N values, where N is the lesser dimension.  We
still
            use all the possible squares in the matrix to perform our
selection,
            but the final score does not consider part of the matrix.

            To perform the work, we figure out which type of matrix we
are
            dealing with (the dimensions).  We use that to select an
array that contains
            the column indexes for each valid combination of segments
(where
            valid means no column participates twice).

            Our matrix always comes either as a square, or as a fat,
short matrix.
            That is, the number of rows is always less than or equal to
the number of
            columns.  This way, we do not have to specify as many
combination arrays,
            since we only have to cover a 2 X 3 array, and not a 3 X 2.

            Also, before this function, we see if either name has just 1
            segment, in which case we use the best score.
*/
double      NH_best_score(int xDim, int yDim, NHSegScoreMode scoreMode,
                                                                double
scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ])
{
      byte  *comboIndexesPtr;             //    points to array that
holds valid column index combos
      int         numCominations;

      switch (xDim)     {
            case  2:
                  switch (yDim)     {
                        case 2:           //    2 by 2
                              comboIndexesPtr = twoByTwo;
                              numCominations = 2;
                              break;
                        case 3:           //    2 by 3
                              comboIndexesPtr = twoByThree;
                              numCominations = 6;
                              break;
                        case 4:           //    2 by 4
                              comboIndexesPtr = twoByFour;
                              numCominations = 12;
                              break;
                        case 5:           //    2 by 5
                              comboIndexesPtr = twoByFive;
                              numCominations = 20;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = twoByFive;
                              numCominations = 20;
                              break;
                  }
                  break;
            case  3:
                  switch (yDim)     {
                        case 3:           //    3 by 3
                              comboIndexesPtr = threeByThree;
                              numCominations = 6;
                              break;
                        case 4:           //    3 by 4
                              comboIndexesPtr = threeByFour;
                              numCominations = 24;
                              break;
                        case 5:           //    3 by 5
                              comboIndexesPtr = threeByFive;
                              numCominations = 60;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = threeByFive;
                              numCominations = 60;
                              break;
                  }
                  break;
            case  4:
                  switch (yDim)     {
                        case 4:           //    4 by 4
                              comboIndexesPtr = fourByFour;
                              numCominations = 24;
                              break;
                        case 5:           //    4 by 5
                              comboIndexesPtr = fourByFive;
                              numCominations = 120;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = fourByFive;
                              numCominations = 120;
                              break;
                  }
                  break;
            case  5:
                  switch (yDim)     {
                        case 5:           //    5 by 5
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
                  }
                  break;
            default:          //    must be greater than 5, so just use
first five
                                                //    also, since xDim
is <= yDim, we do not have to
                                                //    handle 5 X 2, 5 X
3, etc
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
      }

      //    we always use xDim matrix cells to compute the score, since
it
      //    is the smaller of the dimensions.  We go through each
combination
      //    and evaluate the scores found in the scores array for the
      //    particular combination of indexes.
      //    Each evaluation must consider xDim values, so each pass
through the
      //    loop collects xDim values.
      //    The values from the comboIndexesPtr array are the column
indexes.
      //    numCominations is the number of times we iterate through the
loop to
      //    look at a combination of elements in the score matrix.
      //
      //    For example:
      //    if I have a 2 X 3 matrix, I need to find the best valid 2
segment
      //    combination (since 2 is xDim).  There are 6 possible
combinations,
      //    and the column values are stored as pairs in the twoByThree
array.
      //    The row values are implicitly 0 and 1 for each pair, so I
end up
      //    checking:
      //                              scores[0][twoByThree[0]] +
scores[1][twoByThree[1]];
      //                              scores[0][twoByThree[2]] +
scores[1][twoByThree[3]];
      //                              scores[0][twoByThree[4]] +
scores[1][twoByThree[5]];
      //                              scores[0][twoByThree[6]] +
scores[1][twoByThree[7]];
      //                              scores[0][twoByThree[8]] +
scores[1][twoByThree[9]];
      //                              scores[0][twoByThree[10]] +
scores[1][twoByThree[11]];
      //
      double      tempScoreTotal;
      double      tempLowScore;
      double      tempVal;
      double      highestTotal = 0.0;
      double      bestLowScore = 0.0;
      int               comboArrayIndex = 0;
      int               i, row;

      for (i = 0;  i < numCominations; i++)      {
            tempScoreTotal = 0.0;
            tempLowScore = 1.0;
            for (row = 0; row < xDim; row++)     {
                  //    get a single score
                  tempVal =
scores[row][comboIndexesPtr[comboArrayIndex]];
                  //    now see if score is the low score for this combo
                  if (tempVal < tempLowScore)
                        tempLowScore = tempVal;

                  //    include this cell in the total for this
combination
                  tempScoreTotal += tempVal;

                  //    look at next item in the combination (or the
next combination)
                  comboArrayIndex++;
            }
            //    see if the low score is better than our previous low
score
            if (tempLowScore > bestLowScore)
                  bestLowScore = tempLowScore;
            //    see if this score is higher than our previous highest
            if (tempScoreTotal > highestTotal)
                  highestTotal = tempScoreTotal;
      }
      if (scoreMode == NH_SEGMODE_AVG)
            return highestTotal / xDim;
      else
            return bestLowScore;
}


/* NH_best_score_for_highest_mode
            This is a special version of NH_best_score.  For a complete
            description of how the combination stuff works, see the
comments
            for NH_best_score.

            We made this a separate function because:
                  -     it has to return (by reference) an array of
scores.  The other
                        modes only have to return a score for the name.
                  -     The way we figure out which array of scores to
return is
                        much more involved than NH_best_score.
                  -     Since we only do this stuff in highest mode, we
did not
                        want to slow down the processing of
NH_best_score by passing
                        extra parameters and adding lots of "if"
statements.
```

```
            This function was added so that we can figure out which
            combination of segments gives us the highest scores, with
            the highest score being most important, the next highest
score
            being the second most important, etc.  Note that this is
            different from average score, where we are looking for the
            highest sum of scores.  In that case, the higest score is no
            more important that the lowest score.  In fact, the
combination
            chosen in average mode might not even include the single
highest
            segment score.
            To achieve our goal, we evaluate each possible combination
of
            index pairings.  Each combination gives us an array of N
scores,
            where n is the smaller dimension in the matrix.
            We sort each combination so that the highest score appears
first
            in the array, and so on.  If this is the first combination
we
            have evaluated, it becomes the one to beat, so we fill up
the
            array of scores we were passed by reference with this array
of
            scores.  We then go through the rest of the combinations
looking
            for an array that beats the current one to beat.  To beat
it, as
            we walk through the array, we compare the scores from each
array.
            If they are equal, we move on to the next one.  Otherwise,
the
            higher score wins.
            To help speed things up, we get passed in the high score,
which
            is the high score of the entire matrix (note this high score
could
            appear more than once).  We use this high score to quickly
discount
            combinations as not being possible contenders.  If, after
sorting
            a contender array, the first item is not the high score we
were
            passed, this combination could not possibly be the one, so
why
            bother copying all the array elements?
            Note that we check before entering this function to make
sure
            both dimensions are bigger than 1.  And we make sure that
            xdim is the smaller of the dimensions (or they are equal).
*/
void  NH_best_score_for_highest_mode(int xDim, int yDim, double
highestScore,                                                  double

*bestSegScores,
                                                                 double
scores[NH_MAX_SEGS_AFTER_TAQ][NH_MAX_SEGS_AFTER_TAQ])
{
      byte  *comboIndexesPtr;             //    points to array that
holds valid column index combos
      int         numCominations;

      switch (xDim)     {
            case  2:
                  switch (yDim)     {
                        case 2:           //    2 by 2
                              comboIndexesPtr = twoByTwo;
                              numCominations = 2;
                              break;
                        case 3:           //    2 by 3
                              comboIndexesPtr = twoByThree;
                              numCominations = 6;
                              break;
                        case 4:           //    2 by 4
                              comboIndexesPtr = twoByFour;
                              numCominations = 12;
                              break;
                        case 5:           //    2 by 5
                              comboIndexesPtr = twoByFive;
                              numCominations = 20;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = twoByFive;
                              numCominations = 20;
                              break;
                  }
                  break;
            case  3:
                  switch (yDim)     {
                        case 3:           //    3 by 3
                              comboIndexesPtr = threeByThree;
                              numCominations = 6;
                              break;
                        case 4:           //    3 by 4
                              comboIndexesPtr = threeByFour;
                              numCominations = 24;
                              break;
                        case 5:           //    3 by 5
                              comboIndexesPtr = threeByFive;
                              numCominations = 60;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = threeByFive;
                              numCominations = 60;
                              break;
                  }
                  break;
            case  4:
                  switch (yDim)     {
                        case 4:           //    4 by 4
                              comboIndexesPtr = fourByFour;
                              numCominations = 24;
                              break;
                        case 5:           //    4 by 5
```

```
                              comboIndexesPtr = fourByFive;
                              numCominations = 120;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = fourByFive;
                              numCominations = 120;
                              break;
                  }
                  break;
            case  5:
                  switch (yDim)     {
                        case 5:           //    5 by 5
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
                        default:          //    must be greater than 5,
so just use first five
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
                  }
                  break;
            default:          //    must be greater than 5, so just use
first five
                                                //    also, since xDim
is <= yDim, we do not have to
                                                //    handle 5 X 2, 5 X
3, etc
                              comboIndexesPtr = fiveByFive;
                              numCominations = 120;
                              break;
      }

      //    we always use xDim matrix cells to compute the score, since
it
      //    is the smaller of the dimensions.  We go through each
combination
      //    and evaluate the scores found in the scores array for the
      //    particular combination of indexes.
      //    Each evaluation must consider xDim values, so each pass
through the
      //    loop collects xDim values.
      //    The values from the comboIndexesPtr array are the column
indexes.
      //    numCominations is the number of times we iterate through the
loop.to
      //    look at a combination of elements in the score matrix.
      //
      //    For example:
      //    if I have a 2 X 3 matrix, I need to find the best valid 2
segment
      //    combination (since 2 is xDim).  There are 6 possible
combinations,
      //    and the column values are stored as pairs in the twoByThree
array.
      //    The row values are implicitly 0 and 1 for each pair, so I
end up
      //    checking:
      //                                   scores[0][twoByThree[0]] +
scores[1][twoByThree[1]];
```

```
          //                              scores[0][twoByThree[2]] +
scores[1][twoByThree[3]];
          //                              scores[0][twoByThree[4]] +
scores[1][twoByThree[5]];
          //                              scores[0][twoByThree[6]] +
scores[1][twoByThree[7]];
          //                              scores[0][twoByThree[8]] +
scores[1][twoByThree[9]];
          //                              scores[0][twoByThree[10]] +
scores[1][twoByThree[11]];
          //
          double       tempSegScores[NH_MAX_SEGS_AFTER_TAQ];
          int                comboArrayIndex = 0;
          int                i, row;
          bool         includesHighestScore;
          double       swapVal;
          int                tempIndex;
          double       compVal;
          int                numChanges;
          double       tempVal;


          //     init the temp seg scores array to zeros, so that the first
          //     comparison will fail.
          for (tempIndex = 0; tempIndex < xDim; tempIndex++)      {
                bestSegScores[tempIndex] = 0;
          }

          for (i = 0;  i < numCominations; i++)       {
                includesHighestScore = false; //     assume this combo does
not
                                      //     include the best score
                for (row = 0; row < xDim; row++)      {
                      //     get a single score
                      tempVal =
scores[row][comboIndexesPtr[comboArrayIndex]];

                      //     now see if score is the low score for this combo
                      if (tempVal == highestScore)
                            includesHighestScore = true;

                      //     save this value as part of our temp array of
scores
                      //     that we will sort below
                      tempSegScores[row] = tempVal;

                      //     look at next item in the combination (or the
next combination)
                      comboArrayIndex++;
                }
                //     see if this combo includes the best score.   If so,
sort it
                //     and then compare it to the current numbers in
bestSegScores.
                if (includesHighestScore == true)    {
                      //     sort the numbers in bestSegScores
                      while (1)    {
                            numChanges = 0;
                            for (tempIndex = 1; tempIndex < xDim;
tempIndex++)       {
```

```
                              if (tempSegScores[tempIndex - 1] <
tempSegScores[tempIndex])     {
                                    swapVal = tempSegScores[tempIndex -
1];
                                    tempSegScores[tempIndex - 1] =
tempSegScores[tempIndex];
                                    tempSegScores[tempIndex] = swapVal;
                                    numChanges++;
                              }
                        }
                        if (numChanges == 0)
                              break;
                  }

                  //    now compare these temp scores to the current
best scores
                  for (tempIndex = 0; tempIndex < xDim;
tempIndex++)      {
                        compVal = tempSegScores[tempIndex] -
bestSegScores[tempIndex];
                        if (compVal > 0)   {
                              //    temp scores are better, so replace
the best scores with them
                              for (tempIndex = 0; tempIndex < xDim;
tempIndex++)      {
                                    bestSegScores[tempIndex] =
tempSegScores[tempIndex];
                              }
                              break;
                        }
                        else
                              if (compVal < 0)   {
                                    //    current scores are better, so
break out
                                    break;
                              }
                              //    otherwise, just continue the loop.
                  }
            }
      }
}


/* digraph_score
     This is the core of the name-check algorithm.
     A value from 0.0 to 1.0 is calculated based on the number of
     digraphs which match between the two given strings.
     A bias can be used so that digraphs on the right end of the
     strings count less than those on the left.

      Notes:
            The routine ensures that a digraph can only participate in a
            match once.

            Each match results in two points being added to the total.
The
            final score is the total number of points divided by the
number
            of digraphs that could have matched.
            The bias works by discounting the score we award for a
digraph
            match.  As we move into the segment, we subtract 0.1 from
the
            match score.

            The weight table is used to adjust the divisor (which is
normally
            the total number of digraphs that could have matched).  In
the case
            of bias, we need to decrease that number.  Otherwise, an
exact match
            would not return a 1.0, since we would only be deducting
from the
            score (the numerator), and not the divisor.  The weight
table factors
            correspond to the score that would be assigned to an exact
match for
            each possible length.  In other words, we start at 1, then
add .9, then
            add .8, etc.  (the same pattern we use to deduct from the
match score)
*/
double      NH_digraph_score(char *qSeg, int qSegLen,

char *evalSeg, int evalSegLen,
                         bool useLeftDigraphBias)
{
      char tempDigraphStr[2 + 1]; // storage for a digraph string


      //    terminate the temp digraph string once
      tempDigraphStr[2] = EOS;

      //    These are the weights a name has when using a biased
      //    (left-to-right) calculation.  They end up being used as the
denominator
      //    for the final score calculation
      static const double NH_dig_bias_weights[NH_MAX_SEG_LENGTH + 2]
            = { 1.0, 1.0, 1.9, 2.7, 3.4, 4.0, 4.5, 4.9, 5.2, 5.4, 5.5,
5.6, 5.7,
                                          5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4,
6.5, 6.6, 6.7, 6.8, 6.9, 7.0,
                                               7.1, 7.2, 7.3, 7.4, 7.5, 7.6};

      //    an array of 'Y' or 'N' values, one for each possible digraph
      //    position in the eval segment.  Each starts out at 'N' and
gets
      //    to 'Y' when (and if) it gets used.
      //    Note that we must add 1 because we normally pad the name
with
      //    spaces.
      char alreadyMatched[NH_MAX_SEG_LENGTH + 1]; // max digraphs =
NAME_SIZE + 1

      // Forget all previous matches.
      memset(alreadyMatched, 'N', sizeof alreadyMatched);

      // Now count the number of elements involved in matching.
      double      qBiasFactor = 0.9;                 //    0.9 because
```

```
of leading digraph check
        double      evalBiasFactor = 0.9;        //    see note below
        double      matchPoints;
        char        *evalSegString;


        //    start out by checking the first character, which is a
special
        //    case.  It forms an implied digraph of " X" (space, followed
by
        //    the character.  Thus, if both the query and eval have the
same
        //    first character, we give them 2 match points.
        //    Also, since we really start our loop with the second
digraph,
        //    we set the bias factors to 0.9 rather than 1.0
        if (qSeg[0] == evalSeg[0])    {
                matchPoints = 2.0;
        }
        else
                matchPoints = 0.0;

        for (int queryIndex = 0; queryIndex < qSegLen - 1; ++queryIndex) {
                /* see if this digraph occurs in database name */
                tempDigraphStr[0] = qSeg[queryIndex];
                tempDigraphStr[1] = qSeg[queryIndex + 1];
                evalSegString = evalSeg;
                if (useLeftDigraphBias) {
                        //    bring down the query bias by 0.1 each time,
 until we get to 0.1
                        if ((queryIndex > 0) && (queryIndex <  10))
                                qBiasFactor -= 0.1;
                }
                do {
                        evalSegString = strstr(evalSegString, tempDigraphStr);
                        if (evalSegString != NULL) {
                                int evalMatchOffset = evalSegString - evalSeg;

                                if (alreadyMatched[evalMatchOffset] == 'N') {
                                        alreadyMatched[evalMatchOffset] = 'Y';
                                        if (useLeftDigraphBias) { /* decrement
 eval match-bias, minimum 0.10 */
                                                evalBiasFactor = 1.0 - 0.1 *
 (evalMatchOffset + 1);
                                                if (evalBiasFactor < 0.1)
                                                        evalBiasFactor = 0.1;
                                                matchPoints += qBiasFactor +
 evalBiasFactor;
                                        }
                                        else
                                                matchPoints += 2.0;
                                        break;
                                        }
                                else
                                        evalSegString++;
                        }
                } while (evalSegString != NULL);
        }

        //    now do a check for the "hidden" digraph at the end of the
 segment
```

```
      //    to account for the non-existant trailing space
      if (qSeg[qSegLen - 1] == evalSeg[evalSegLen - 1])     {
            if (useLeftDigraphBias) {
                  evalBiasFactor = 1.0 - 0.1 * evalSegLen;
                  if (evalBiasFactor < 0.1)
                        evalBiasFactor = 0.1;
                  //    don't forget to bring down the query bias by 0.1
also,
                  //    unless we are at 0.1
                  if ((queryIndex > 0) && (queryIndex <  10))
                        qBiasFactor -= 0.1;
                  matchPoints += qBiasFactor + evalBiasFactor;
            }
            else
                  matchPoints += 2.0;
      }


      // The return value is the number of elements involved in matching
      // compared to the total number of elements.
      return useLeftDigraphBias
                               ? matchPoints /
(NH_dig_bias_weights[qSegLen + 1] + NH_dig_bias_weights[evalSegLen + 1])
                               : matchPoints / (qSegLen + evalSegLen +
2);
} /* NH_digraph_score */



/*
      This function adjusts the diScore (which already has some value)
based
      on the TAQ values that are attached to the two segments passed in.

      In the NameHunter system, TAQs are broken up into two types
(disregard and
      delete).  In general, disregard TAQs (e.g. "Jr.") contain more
      meaningful information than delete TAQs (e.g. "Mr."), and thus
      disregard TAQs are considered more important when
evaluating/comparing
      TAQs between segments.

      There are three factors involved in modifying the score.  These
are

            -     delete factor
            -     disregard factor
            -     absent factor

      When applied, a factor is multiplied by the existing score.
However,
      deciding which factor (if any) to apply is somewhat complex,
especially
      when one or both of the segments have multiple TAQ values.  For
this
      reason, we describe the multi-TAQ situation separately.

      For situations where both segments have either 0 or 1 TAQ values,
we
      use the following matrix to choose a factor to apply:
```

```
                                                       |           No
TAQ    |          Delete TAQ          |           Disregard
TAQ          |
            --------------|-----------|-----------------|---------------
----|
            No TAQ                         |     No Change   |     Absent
Factor             |    Absent Factor                  |
                                                       |
                   |
|                                                            |
            --------------|-----------|-----------------|---------------
----|
            Delete       TAQ  |     Absent             |Delete
Factor             |    Absent Factor                  |
                                                       |     Factor
       |Unless
same               |
       |
            --------------|-----------|-----------------|---------------
----|
            Disregard TAQ     |     Absent             |Disregard
Factor       |     Absent
Factor                   |
             |
                                                       |     Factor
       |Unless
same               |
       |                                                           |
            --------------|-----------|-----------------|---------------
----|
```

For the multiple case, we use the algorithm below. A general word about the alg - we are treating disregard as more important than delete, so we start out by checking for disregards. All it takes is for one disregard value in each of the segments to match to avoid applying the disregard factor. The same goes for deletes. If we have any dis values in one segment, but none in the other, we apply the absent factor.

Assuming segments S1 and S2:

```
      -     Look for dis segments in S1
      -     if found
            -     if same segment found in S2
                  -     go on to delete processing
            -     else
                  -     if no dis segments in S2
                        -     apply absent value
                  -     else, continue looking for dis segments in
S1 that match S2
                        if we get to end of S1 segments and still
have not found a
                        matching dis in S2, apply dis factor.
      -     else  (no dis found in S1)
            -     look for dis in S2
                  -     if found
                        -     apply absent
                        -     else
                              -     go on to delete processing

      Delete processing:

      -     look for deletes in S1
            -     if found
                  -     if same seg found in S2
                        -     do nothing
                  -     else
                        -     if no deletes in S2
                              -     apply absent
                        -     else
                              -     continue to look for deletes in S1.
If we get to end if

                                    S1 segments and do not find any

deletes that match a
                                    delete in S2, apply delete factor

            -     else  (not deletes found in S1)
                  -     look for deletes in S2
                        -     if delete found
                              -     apply absent
                        -     else
                              -     do nothing.
*/
void  NH_apply_TAQs_to_score(double *diScore, Segment *qSeg, Segment
*evalSeg,

                        double absDelTAQFactor,

                        double absDisTAQFactor,

                        double delTAQFactor,

                        double disTAQFactor)
{
      int               numQTAQs = qSeg->numTAQs;
      int               numEvalTAQs = evalSeg->numTAQs;
      double      applyFactor = 1.0;


      //    handle the simple case first
      if ((numQTAQs <= 1) && (numEvalTAQs <= 1))      {
            switch      (numQTAQs)  {
                  case  0:
                        if (numEvalTAQs == 1)   {
                              if (evalSeg->taqList[0].taqAction ==
NH_TAQ_ACTION_DELETE)
                                    applyFactor = absDelTAQFactor;
                              else
                                    applyFactor = absDisTAQFactor;
                        }
                        break;
                  case  1:
                        if (numEvalTAQs == 1)   {
                              //    both segs have 1 TAQ value, so
                              //    figure out the type of TAQs involved
                              if (qSeg->taqList[0].taqAction ==
NH_TAQ_ACTION_DELETE)   {
                                    if (evalSeg->taqList[0].taqAction ==
NH_TAQ_ACTION_DELETE)   {
                                          //    same action, so see if
string are the same
                                          if (strcmp(qSeg-
>taqList[0].segString,
      evalSeg->taqList[0].segString))
                                                applyFactor =
delTAQFactor;                 //    they were different, so apply delete
factor
                                    }
                                    else        //    not the same
action, so do the absent
                                          applyFactor = absDisTAQFactor;
                              }
                              else  {           //    not
NH_TAQ_ACTION_DELETE, so must be
                                                             //    disreg
ard
                                    if (evalSeg->taqList[0].taqAction ==
NH_TAQ_ACTION_DISREGARD)       {
                                          //    same action, so see if
string are the same
                                          if (strcmp(qSeg-
>taqList[0].segString,
      evalSeg->taqList[0].segString))
                                                applyFactor =
disTAQFactor;                 //    they were different, so apply dis
factor
                                    }
                                    else        //    not the same
action, so do the absent dis
                                          applyFactor =
absDisTAQFactor;  //    since dis takes precidence of del
                              }
                        }
                        else  {     //    query had 1 TAQ, but eval had
none
                              if (qSeg->taqList[0].taqAction ==
NH_TAQ_ACTION_DELETE)
                                    applyFactor = absDelTAQFactor;
                              else
                                    applyFactor = absDisTAQFactor;
                        }
                        break;
            }
      }
      else  {
            //    one (or both) of the segments has more than 1 TAQ
value

            //    First see if either has no TAQ segments.  In this
case,
            //    we can apply the absent factor and skip the ugly
processing
            //    below
            if (numQTAQs == 0)      {
                  //    assume the abs del factor, but look for a DIS in
the
                  //    eval.  If we find one, set the applyFactor to
```

```
the abs dis
                  //    since that should take precidence
                  applyFactor = absDelTAQFactor;
                  for   (int evalIndex = 0; evalIndex < numEvalTAQs;
evalIndex++)      {
                        if (evalSeg->taqList[evalIndex].taqAction ==
NH_TAQ_ACTION_DISREGARD)      {
                              applyFactor = absDisTAQFactor;
                              break;
                        }
                  }
            }
            else if (numEvalTAQs == 0)    {
                  //    assume the abs del factor, but look for a DIS in
the
                  //    query.  If we find one, set the applyFactor to
the abs dis
                  //    since that should take precidence
                  applyFactor = absDelTAQFactor;
                  for   (int qIndex = 0; qIndex < numQTAQs;
qIndex++)   {
                        if (qSeg->taqList[qIndex].taqAction ==
NH_TAQ_ACTION_DISREGARD)      {
                              applyFactor = absDisTAQFactor;
                              break;
                        }
                  }
            }
            else  {
                  //    one segment has 2 or more TAQs, and the other
has one or more

                  bool  satisfiedDis = true;    //    we assume we have
satified the

                              //    dis processing until we find

                              //    a dis value, since if neither

                              //    seg has a dis value, we do not

                              //    apply the dis value
                  bool  satisfiedDel = true;    //    we assume we have
satified the

                              //    del processing until we find

                              //    a del value, since if neither

                              //    seg has a del value, we do not

                              //    apply the del value
                  bool  satisfiedAbs = true;    //    we assume we have
satified the

                              //    abs processing.
                  bool  foundMatchingDis = false;
                  bool  foundMatchingDel = false;


                  int   i, j;
```

```
                    //    go through the query segment, looking for dis
segments
                    for   (i = 0; i < numQTAQs; i++)    {
                          if (qSeg->taqList[i].taqAction ==
NH_TAQ_ACTION_DISREGARD)        {
                                //    since we found a dis, we must find a
dis in the eval seg.
                                satisfiedDis = false;
                                satisfiedAbs = false;
                                //    look for disregards in the eval seg.
                                for   (j = 0; j < numEvalTAQs; j++) {
                                      if (evalSeg->taqList[j].taqAction ==
NH_TAQ_ACTION_DISREGARD)        {
                                            //    found a dis, so we are
not dealing with an absent
                                            //    situation - see if the
segs are the same.
                                            satisfiedAbs = true;
                                            if (!strcmp(qSeg-
>taqList[i].segString,
      evalSeg->taqList[j].segString))       {
                                                  foundMatchingDis = true;
                                                  satisfiedDis = true;
                                                  break;
                                            }
                                      }
                                }
                                //    if we get here, and the abs has not
been satified, we
                                //    apply the abs factor, since we did
not find any dis in the
                                //    eval, but did find one in the query.
                                if (satisfiedAbs == false)    {
                                      applyFactor = absDisTAQFactor;
                                      //    mark the DIS as satisfied so
that we do not
                                      //    re-assign the factor below
when seeing if DEL was satisfied.
                                      satisfiedDis = true;
                                      break;
                                }
                                else  {
                                      //    check to see if we satisfied
the dis.  If we did, we can
                                      //    go check out the delete stuff.
                                      if (satisfiedDis == true)
                                            break;
                                }
                          }
                    }                 //    end for query TAQ

                    //    once here, we made it to the end of the query
TAQs while looking
                    //    for disregards., This means either:
                    //          -     we found no disregards in the query
- so go on
                    //                      and see if there are any
```

```
disregards in the Eval
                  //          -     we found disregards in Q, but none
in Eval - we
                  //                      apply the absent factor, and
we're done
                  //          -     we found dis in Q, but no matching
ones in Eval - we
                  //                      apply the disregard factor,
and we're done
                  //          -     we found a matching dis in Q and
Eval  -     so do deletes.
                  //                      we can skip the check for
disregards in Eval, since
                  //                      we already know there is a
match.

                  //    make sure we should continue
                  if (satisfiedAbs && satisfiedDis)   {
                        //
                        if (foundMatchingDis == false)      {
                              //    We are in this section if the Q had
no Dis Values.
                              //    see if there are dis values in Eval.
                              for   (j = 0; j < numEvalTAQs; j++) {
                                    if (evalSeg->taqList[j].taqAction ==
NH_TAQ_ACTION_DISREGARD)      {
                                          applyFactor = absDisTAQFactor;
                                          satisfiedAbs = false;
                                          break;
                                    }
                              }
                        }

                        //    see if we should still continue after
checking for reverse absent
                        if (satisfiedAbs) {
                              //    when here, we got passed checking
for the dis, so we need to check for
                              //    deletes.

                              //    go through the query segment,
looking for del segments
                              for   (i = 0; i < numQTAQs; i++)    {
                                    if (qSeg->taqList[i].taqAction ==
NH_TAQ_ACTION_DELETE)   {
                                          //    since we found a del, we
must find a del in the eval seg.
                                          satisfiedDel = false;
                                          satisfiedAbs = false;
                                          //    look for deletes in the
eval seg.
                                          for   (j = 0; j < numEvalTAQs;
j++)  {
                                                if (evalSeg-
>taqList[j].taqAction == NH_TAQ_ACTION_DELETE)  {
                                                      //    found a del,
so we are not dealing with an absent
                                                      //    situation -
see if the segs are the same.
                                                      satisfiedAbs =
true;
```

```
                                                      if (!strcmp(qSeg-
>taqList[i].segString,
                  eval~~Seg~~->taqList[j].segString))     {
                                                            foundMatchin
gDel = true;
                                                            satisfiedDel
= true;
                                                            break;
                                                      }
                                                }
                                          }

                                          //    if we get here, and the
abs has not been satified, we
                                          //    apply the abs factor,
since we did not find any del in the
                                          //    eval, but did find one
in the query.
                                          if (satisfiedAbs ==
false)      {
                                                applyFactor =
absDelTAQFactor;
                                                //    mark the DEL as
satisfied so that we do not
                                                //    re-assign the
factor below when seeing if DEL was satisfied.
                                                satisfiedDel =
true;
                                                break;
                                          }
                                          else  {
                                                //    check to see if we
satisfied the del.  If we did, were done
                                                if (satisfiedDel ==
true)
                                                      break;
                                          }
                                    }
                              }                 //    end for query TAQ

                              //    make sure we should continue
                              if (satisfiedAbs && satisfiedDel)   {
                                    if (foundMatchingDel ==
false)      {
                                          //    We are in this section
if the Q had no Del Values.
                                          //    see if there are del
values in Eval.
                                          for   (j = 0; j < numEvalTAQs;
j++)  {
                                                if (evalSeg-
>taqList[j].taqAction == NH_TAQ_ACTION_DELETE)  {
                                                      applyFactor =
absDelTAQFactor;
                                                      satisfiedAbs =
false;
                                                      break;
                                                }
                                          }
                                    }
                              }

                        }
                  }

                  //    decide the factor based on the condition that
was not satisfied
                  //    except for abs, in which case we already set the
applyFactor
                  //    above
                  if (satisfiedDel == false)
                        applyFactor = delTAQFactor;
                  else  if (satisfiedDis == false)
                        applyFactor = disTAQFactor;

            }
      }

      //    apply the factor we decided on
      *diScore *= applyFactor;
}
```

```
// DigraphBitmapArray.hpp : header file
//
//    Class that holds the bit patterns for each possible
//    digraph (AA - ZZ).  We also need to account for spaces.
//
//    Each bit pattern turns on just 1 bit.  We basically turn
//    on one bit, and shift it through the value until it reaches
//    the other end, at which time we start back at the beginning
//    again.
//
//    Any other character are treated as spaces in our scheme,
//    so we do not need to worry about them.
//
//    The class supports either a 32 bit value, or a 64 bit value.
////////////////////////////////////////////////////////////////////////////
/////

#ifndef DIGRAPHBITMAPARRAY_HPP
#define DIGRAPHBITMAPARRAY_HPP


//    How many indexes do we need in our two dimensional array?
//    27 (26 letters plus a space)
#define          BITMAP_ARRAY_INDEX_SIZE       27


typedef          struct      {
      unsigned    int   hiBytes;
      unsigned    int   lowBytes;
}           bit_64_t;


class NHDigraphBitmapArray
{
// Construction
public:
      NHDigraphBitmapArray(); // standard constructor

      ~NHDigraphBitmapArray();

      unsigned int      get32BitKeyForToken(char *token);

      void                          get64BitKeyForToken(char *token,
bit_64_t *key);

      unsigned char     getNumBitsForByte(unsigned char byteVal) {return
bitTable[byteVal];}

// Implementation
protected:

      void  buildBitTable();

      //    the array that holds the bit map paterns for each possible
      //    digraph.  Each item in the array is an integer that has
      //    one of its 32 bits turned on.
      unsigned
int         bitMapArray32[BITMAP_ARRAY_INDEX_SIZE][BITMAP_ARRAY_INDEX_SI
ZE];

      //    the array that holds the bit map paterns for each possible
      //    digraph.  Each item in the array is an integer that has
      //    one of its 64 bits turned on.
      bit_64_t                      bitMapArray64[BITMAP_ARRAY_INDEX_SIZ
E][BITMAP_ARRAY_INDEX_SIZE];

      unsigned char           bitTable[256];

};


#endif
```

```
// NHDigraphBitmapArray.cpp : implementation file
//
//          3/20/98     EFB         Changed names to NH from SN

#include "NHDigraphBitmapArray.hpp"

#include    <stdio.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


typedef     unsigned char byte;


//////////////////////////////////////////////////////////////////////////
/////
//    Constructor.
//    Fills in the values in both of the bitMapArrays (32 bit and
//    64 bits).
NHDigraphBitmapArray::NHDigraphBitmapArray()
{
      unsigned int      bitmapValue32 = 1;
      unsigned int      bitmapValue64High = 0;
      unsigned int      bitmapValue64Low = 1;

      for   (int i = 0; i < BITMAP_ARRAY_INDEX_SIZE; i++)    {
            for   (int j = 0; j < BITMAP_ARRAY_INDEX_SIZE; j++)    {

                  //    assign the 32 bit value
                  bitMapArray32[i][j] = bitmapValue32;

                  //    assign the 64 bit value
                  bitMapArray64[i][j].hiBytes = bitmapValue64High;
                  bitMapArray64[i][j].lowBytes = bitmapValue64Low;

                  //    now shift the values
                  bitmapValue32 <<= 1;
                  if (bitmapValue32 == 0)
                        bitmapValue32 = 1;

                  if (bitmapValue64Low == 0)    {
                        bitmapValue64High <<= 1;
                        if (bitmapValue64High == 0)    {
                              bitmapValue64Low = 1;
                        }
                  }
                  else  {
                        bitmapValue64Low <<= 1;
                        if (bitmapValue64Low == 0)    {
                              bitmapValue64High = 1;
                        }
                  }
            }
      }

      buildBitTable();
}
```

```
NHDigraphBitmapArray::~NHDigraphBitmapArray()
{
}


void  NHDigraphBitmapArray::get64BitKeyForToken(char *token, bit_64_t
*key)
{
      char                                *ch1;
      char                                *ch2;
      int                                       index1;
      int                                       index2;
      char                                spacedToken[200];

      //    zero out the key we are going to return
      key->hiBytes = 0;
      key->lowBytes = 0;

      sprintf(spacedToken, " %s ", token);

      ch1 = spacedToken;
      if (*ch1 != '\0') {
            ch2 = ch1 + 1;
            while (*ch2 != '\0')    {
                  if (*ch1 == ' ')
                        index1 = 26;
                  else
                        index1 = *ch1 - 'A';
                  if (*ch2 == ' ')
                        index2 = 26;
                  else
                        index2 = *ch1 - 'A';
                  if ((index1 >= 0) && (index1 <
BITMAP_ARRAY_INDEX_SIZE)
                              &&  (index2 >= 0) && (index2 <
BITMAP_ARRAY_INDEX_SIZE))     {
                        key->hiBytes |=
bitMapArray64[index1][index2].hiBytes;
                        key->lowBytes |=
bitMapArray64[index1][index2].lowBytes;
                  }
                  ch1 = ch2;
                  ch2++;
            }
      }

}


unsigned int      NHDigraphBitmapArray::get32BitKeyForToken(char *token)
{
      unsigned int            retVal = 0;
      char                                *ch1;
      char                                *ch2;
      int                                       index1;
      int                                       index2;
      char                                spacedToken[200];
      sprintf(spacedToken, " %s ", token);

      ch1 = spacedToken;
      if (*ch1 != '\0') {
            ch2 = ch1 + 1;
            while (*ch2 != '\0')    {
                  if (*ch1 == ' ')
                        index1 = 26;
                  else
                        index1 = *ch1 - 'A';
                  if (*ch2 == ' ')
                        index2 = 26;
                  else
                        index2 = *ch1 - 'A';
                  if ((index1 >= 0) && (index1 <
BITMAP_ARRAY_INDEX_SIZE)
                              &&  (index2 >= 0) && (index2 <
BITMAP_ARRAY_INDEX_SIZE))
                        retVal |= bitMapArray32[index1][index2];
                  ch1 = ch2;
                  ch2++;
            }
      }

      return retVal;
}


//    build a table that says how many bits a byte value
//    has turned off.
void  NHDigraphBitmapArray::buildBitTable()
{
      byte  tempByte;
      int         i, j;
      byte  bitsTurnedOff;

      for (i = 0; i < 256; i++)     {
            tempByte = i;
            bitsTurnedOff = 0;
            for (j = 0; j < 8; j++) {
                  if (tempByte & 1)                   //    use this
when array says how many 1's
                  //    if ((tempByte & 1) == 0)            //    use
this when array says how many 0's
                        bitsTurnedOff++;
                  tempByte >>= 1;
            }
            bitTable[i] = bitsTurnedOff;
      }
}
```

```
//      File:  NHCompParms.cpp
//
//      Description:
//
//              Implementation to the NHCompParms class.
//
//
//      History:
//
//              5/8/97          EFB             Created
//              3/3/98          EFB             Changed name of class, and move PP
parms to
//                                                                      the new
NHNameParms class.
//              3/20/98         EFB             Changed names to NH from SN
//


#include <string.h>
#include <stdio.h>
#include <stdlib.h>


#include        "NHCompParms.hpp"
#include        "NHVariantTable.hpp"
#include        "NHTAQTable.hpp"
#include        "NH_variant_taq_globals.h"


NHCompParms::NHCompParms(NHParmsType parmsType)
{
        status = NH_SUCCESS;


        switch (parmsType)      {

                case  NH_PARMS_GENERIC:                         //      default
                      scoreThresh = 0.6;
                      useGnLeftBias = false;
                      useSnLeftBias = false;
                      matchGnIntial = true;
                      matchSnIntial = false;
                      gnInitialScore = 0.85;
                      snInitialScore = 0.0;
                      gnInitialOnInitialMatchScore = 1.0;
                      snInitialOnInitialMatchScore = 0.0;
                      useGnVariants = true;
                      useSnVariants = true;
                      fnuScore = 0.60;
                      nfnScore = 0.65;
                      lnuScore = 0.6;
                      nlnScore = 0.65;
                      gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
                      snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
                      gnAnchorFactor = 0.0;
                      snAnchorFactor = 0.0;
                      gnOOPSFactor = 0.6;
                      snOOPSFactor = 0.6;
                      disGnTAQFactor = 0.7;
                      absDelGnTAQFactor = 0.9;
```

```
      absDisGnTAQFactor = 0.8;
      delGnTAQFactor = 0.85;
      disSnTAQFactor = 0.7;
      absDelSnTAQFactor = 0.9;
      absDisSnTAQFactor = 0.8;
      delSnTAQFactor = 0.85;
      checkGnCompressedName = false;
      checkSnCompressedName = false;
      gnCompressedNameScore = 0.0;
      snCompressedNameScore = 0.0;
      scoreGnTaqs = true;
      scoreSnTaqs = true;
      gnSegmentScoreMode = NH_SEGMODE_AVG;
      snSegmentScoreMode = NH_SEGMODE_AVG;
      gnScoreThresh = 0.5;
      snScoreThresh = 0.5;
      gnWeight = 0.8;
      snWeight = 1.0;
      break;

case  NH_PARMS_ANGLO:
      scoreThresh = 0.6;
      useGnLeftBias = false;
      useSnLeftBias = false;
      matchGnIntial = true;
      matchSnIntial = false;
      gnInitialScore = 0.85;
      snInitialScore = 0.0;
      gnInitialOnInitialMatchScore = 1.0;
      snInitialOnInitialMatchScore = 0.0;
      useGnVariants = true;
      useSnVariants = true;
      fnuScore = 0.60;
      nfnScore = 0.65;
      lnuScore = 0.6;
      nlnScore = 0.65;
      gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
      snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
      gnAnchorFactor = 0.0;
      snAnchorFactor = 0.0;
      gnOOPSFactor = 0.6;
      snOOPSFactor = 0.6;
      disGnTAQFactor = 0.7;
      absDelGnTAQFactor = 0.9;
      absDisGnTAQFactor = 0.8;
      delGnTAQFactor = 0.85;
      disSnTAQFactor = 0.7;
      absDelSnTAQFactor = 0.9;
      absDisSnTAQFactor = 0.8;
      delSnTAQFactor = 0.85;
      checkGnCompressedName = false;
      checkSnCompressedName = false;
      gnCompressedNameScore = 0.0;
      snCompressedNameScore = 0.0;
      scoreGnTaqs = true;
      scoreSnTaqs = true;
      gnSegmentScoreMode = NH_SEGMODE_AVG;
      snSegmentScoreMode = NH_SEGMODE_AVG;
      gnScoreThresh = 0.5;
      snScoreThresh = 0.5;
      gnWeight = 0.8;
```

```
         snWeight = 1.0;
         break;

case     NH_PARMS_ARABIC:
         scoreThresh = 0.63;
         useGnLeftBias = false;
         useSnLeftBias = false;
         matchGnIntial = true;
         matchSnIntial = true;
         gnInitialScore = 0.85;
         snInitialScore = 0.85;
         gnInitialOnInitialMatchScore = 1.0;
         snInitialOnInitialMatchScore = 1.0;
         useGnVariants = false;
         useSnVariants = false;
         fnuScore = 0.60;
         nfnScore = 0.65;
         lnuScore = 0.6;
         nlnScore = 0.65;
         gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
         snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
         gnAnchorFactor = 0.0;
         snAnchorFactor = 0.0;
         gnOOPSFactor = 0.7;
         snOOPSFactor = 0.9;
         disGnTAQFactor = 0.7;
         absDelGnTAQFactor = 0.9;
         absDisGnTAQFactor = 0.8;
         delGnTAQFactor = 0.85;
         disSnTAQFactor = 0.7;
         absDelSnTAQFactor = 0.9;
         absDisSnTAQFactor = 0.8;
         delSnTAQFactor = 0.85;
         checkGnCompressedName = true;
         checkSnCompressedName = true;
         gnCompressedNameScore = 0.9;
         snCompressedNameScore = 0.9;
         scoreGnTaqs = true;
         scoreSnTaqs = true;
         gnSegmentScoreMode = NH_SEGMODE_AVG;
         snSegmentScoreMode = NH_SEGMODE_AVG;
         gnScoreThresh = 0.63;
         snScoreThresh = 0.63;
         gnWeight = 1.0;
         snWeight = 0.8;
         break;

case     NH_PARMS_CHINESE:
         scoreThresh = 0.70;
         useGnLeftBias = false;
         useSnLeftBias = false;
         matchGnIntial = false;
         matchSnIntial = false;
         gnInitialScore = 0.0;
         snInitialScore = 0.0;
         gnInitialOnInitialMatchScore = 0.0;
         snInitialOnInitialMatchScore = 0.0;
         useGnVariants = true;
         useSnVariants = true;
         fnuScore = 0.60;
         nfnScore = 0.65;
```

```
      lnuScore = 0.6;
      nlnScore = 0.65;
      gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
      snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
      gnAnchorFactor = 0.0;
      snAnchorFactor = 0.0;
      gnOOPSFactor = 0.0;
      snOOPSFactor = 1.0;
      disGnTAQFactor = 0.7;
      absDelGnTAQFactor = 0.9;
      absDisGnTAQFactor = 0.8;
      delGnTAQFactor = 0.85;
      disSnTAQFactor = 0.7;
      absDelSnTAQFactor = 0.9;
      absDisSnTAQFactor = 0.8;
      delSnTAQFactor = 0.85;
      checkGnCompressedName = false;
      checkSnCompressedName = false;
      gnCompressedNameScore = 0.0;
      snCompressedNameScore = 0.0;
      scoreGnTaqs = true;
      scoreSnTaqs = true;
      gnSegmentScoreMode = NH_SEGMODE_LOWEST;
      snSegmentScoreMode = NH_SEGMODE_AVG;
      gnScoreThresh = 0.7;
      snScoreThresh = 0.7;
      gnWeight = 0.8;
      snWeight = 1.0;
      break;

case  NH_PARMS_HISPANIC:
      scoreThresh = 0.60;
      useGnLeftBias = false;
      useSnLeftBias = false;
      matchGnIntial = true;
      matchSnIntial = true;
      gnInitialScore = 0.85;
      snInitialScore = 0.85;
      gnInitialOnInitialMatchScore = 1.0;
      snInitialOnInitialMatchScore = 1.0;
      useGnVariants = true;
      useSnVariants = true;
      fnuScore = 0.60;
      nfnScore = 0.65;
      lnuScore = 0.6;
      nlnScore = 0.65;
      gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
      snAnchorSegmentMode = NH_ANCHOR_SEG_FIRST;
      gnAnchorFactor = 0.0;
      snAnchorFactor = 0.70;
      gnOOPSFactor = 0.6;
      snOOPSFactor = 0.6;
      disGnTAQFactor = 0.7;
      absDelGnTAQFactor = 0.9;
      absDisGnTAQFactor = 0.8;
      delGnTAQFactor = 0.85;
      disSnTAQFactor = 0.7;
      absDelSnTAQFactor = 0.9;
      absDisSnTAQFactor = 0.8;
      delSnTAQFactor = 0.85;
      checkGnCompressedName = true;
```

```
                    checkSnCompressedName = true;
                    gnCompressedNameScore = 0.9;
                    snCompressedNameScore = 0.9;
                    scoreGnTaqs = true;
                    scoreSnTaqs = true;
                    gnSegmentScoreMode = NH_SEGMODE_AVG;
                    snSegmentScoreMode = NH_SEGMODE_AVG;
                    gnScoreThresh = 0.6;
                    snScoreThresh = 0.6;
                    gnWeight = 0.8;
                    snWeight = 1.0;
                    break;

               case  NH_PARMS_KOREAN:                        // Parameters
tuned for Korean names.
                    scoreThresh = 0.66;
                    useGnLeftBias = false;
                    useSnLeftBias = false;
                    matchGnIntial = false;
                    matchSnIntial = false;
                    gnInitialScore = 0.0;
                    snInitialScore = 0.0;
                    gnInitialOnInitialMatchScore = 0.0;
                    snInitialOnInitialMatchScore = 0.0;
                    useGnVariants = true;
                    useSnVariants = true;
                    fnuScore = 0.60;
                    nfnScore = 0.65;
                    lnuScore = 0.6;
                    nlnScore = 0.65;
                    gnAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
                    snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
                    gnAnchorFactor = 0.0;
                    snAnchorFactor = 0.0;
                    gnOOPSFactor = 0.69;
                    snOOPSFactor = 0.63;
                    disGnTAQFactor = 0.7;
                    absDelGnTAQFactor = 0.9;
                    absDisGnTAQFactor = 0.8;
                    delGnTAQFactor = 0.85;
                    disSnTAQFactor = 0.7;
                    absDelSnTAQFactor = 0.9;
                    absDisSnTAQFactor = 0.8;
                    delSnTAQFactor = 0.85;
                    checkGnCompressedName = false;
                    checkSnCompressedName = false;
                    gnCompressedNameScore = 0.0;
                    snCompressedNameScore = 0.0;
                    scoreGnTaqs = true;
                    scoreSnTaqs = true;
                    gnSegmentScoreMode = NH_SEGMODE_AVG;
                    snSegmentScoreMode = NH_SEGMODE_AVG;
                    gnScoreThresh = 0.69;
                    snScoreThresh = 0.63;
                    gnWeight = 0.8;
                    snWeight = 1.0;
                    break;

               case  NH_PARMS_RUSSIAN:                       // Parameters
tuned for Russian names.
                    scoreThresh = 0.61;
```

```
            useGnLeftBias = false;
            useSnLeftBias = true;
            matchGnIntial = true;
            matchSnIntial = true;
            gnInitialScore = 0.85;
            snInitialScore = 0.85;
            gnInitialOnInitialMatchScore = 1.0;
            snInitialOnInitialMatchScore = 1.0;
            useGnVariants = false;
            useSnVariants = false;
            fnuScore = 0.60;
            nfnScore = 0.65;
            lnuScore = 0.6;
            nlnScore = 0.65;
            gnAnchorSegmentMode = NH_ANCHOR_SEG_FIRST;
            snAnchorSegmentMode = NH_ANCHOR_SEG_NONE;
            gnAnchorFactor = 0.60;
            snAnchorFactor = 0.00;
            gnOOPSFactor = 0.65;
            snOOPSFactor = 0.8;
            disGnTAQFactor = 0.7;
            absDelGnTAQFactor = 0.9;
            absDisGnTAQFactor = 0.8;
            delGnTAQFactor = 0.85;
            disSnTAQFactor = 0.7;
            absDelSnTAQFactor = 0.9;
            absDisSnTAQFactor = 0.8;
            delSnTAQFactor = 0.85;
            checkGnCompressedName = false;
            checkSnCompressedName = false;
            gnCompressedNameScore = 0.0;
            snCompressedNameScore = 0.0;
            gnSegmentScoreMode = NH_SEGMODE_HIGHEST;
            snSegmentScoreMode = NH_SEGMODE_AVG;
            gnScoreThresh = 0.6;
            snScoreThresh = 0.62;
            gnWeight = 0.8;
            snWeight = 1.0;
            break;
    }       //    end of switch
}

NHCompParms::NHCompParms(istream &inStream)
{
    int    compParmsVersion;

    if (inStream.good())    {
        inStream.read((char *)&compParmsVersion, sizeof(int));

        inStream.read((char *)&scoreThresh, sizeof(double));
        inStream.read((char *)&useGnLeftBias, sizeof(bool));
        inStream.read((char *)&useSnLeftBias, sizeof(bool));
        inStream.read((char *)&matchGnIntial, sizeof(bool));
        inStream.read((char *)&matchSnIntial, sizeof(bool));
        inStream.read((char *)&gnInitialScore, sizeof(double));
        inStream.read((char *)&snInitialScore, sizeof(double));
        inStream.read((char *)&useGnVariants, sizeof(bool));
        inStream.read((char *)&useSnVariants, sizeof(bool));
        inStream.read((char *)&fnuScore, sizeof(double));
        inStream.read((char *)&nfnScore, sizeof(double));
```

```
            inStream.read((char *)&lnuScore, sizeof(double));
            inStream.read((char *)&nlnScore, sizeof(double));

            inStream.read((char *)&gnSegmentScoreMode,
sizeof(NHSegScoreMode));
            inStream.read((char *)&snSegmentScoreMode,
sizeof(NHSegScoreMode));
            inStream.read((char *)&gnAnchorSegmentMode,
sizeof(NHAnchorSegMode));
            inStream.read((char *)&snAnchorSegmentMode,
sizeof(NHAnchorSegMode));

            inStream.read((char *)&gnAnchorFactor, sizeof(double));
            inStream.read((char *)&snAnchorFactor, sizeof(double));
            inStream.read((char *)&gnOOPSFactor, sizeof(double));
            inStream.read((char *)&snOOPSFactor, sizeof(double));

            inStream.read((char *)&scoreGnTaqs, sizeof(bool));
            inStream.read((char *)&scoreSnTaqs, sizeof(bool));

            inStream.read((char *)&absDelGnTAQFactor, sizeof(double));
            inStream.read((char *)&absDisGnTAQFactor, sizeof(double));
            inStream.read((char *)&absDelSnTAQFactor, sizeof(double));
            inStream.read((char *)&absDisSnTAQFactor, sizeof(double));
            inStream.read((char *)&delGnTAQFactor, sizeof(double));
            inStream.read((char *)&delSnTAQFactor, sizeof(double));
            inStream.read((char *)&disGnTAQFactor, sizeof(double));
            inStream.read((char *)&disSnTAQFactor, sizeof(double));

            inStream.read((char *)&checkGnCompressedName, sizeof(bool));
            inStream.read((char *)&checkSnCompressedName, sizeof(bool));

            inStream.read((char *)&gnCompressedNameScore,
sizeof(double));
            inStream.read((char *)&snCompressedNameScore,
sizeof(double));

            inStream.read((char *)&gnScoreThresh, sizeof(double));
            inStream.read((char *)&snScoreThresh, sizeof(double));

            inStream.read((char *)&gnWeight, sizeof(double));
            inStream.read((char *)&snWeight, sizeof(double));

            inStream.read((char *)&gnInitialOnInitialMatchScore,
sizeof(double));
            inStream.read((char *)&snInitialOnInitialMatchScore,
sizeof(double));

            status = NH_SUCCESS;
      }
      else
            status = NH_COMP_PARMS_BAD_STREAM_ON_CONSTRUCT;
}


NHCompParms::~NHCompParms()
{
}


NHReturnCode      NHCompParms::archiveData(ostream &outStream)
```

```
{
//     comp parms file version history
//           1.0         -     first version
      int                                    compParmsVersion = 1;
      NHReturnCode      rc = NH_SUCCESS;

      if (outStream.good())    {
            outStream.write((char *)&compParmsVersion, sizeof(int));

            outStream.write((char *)&scoreThresh, sizeof(double));
            outStream.write((char *)&useGnLeftBias, sizeof(bool));
            outStream.write((char *)&useSnLeftBias, sizeof(bool));
            outStream.write((char *)&matchGnIntial, sizeof(bool));
            outStream.write((char *)&matchSnIntial, sizeof(bool));
            outStream.write((char *)&gnInitialScore, sizeof(double));
            outStream.write((char *)&snInitialScore, sizeof(double));
            outStream.write((char *)&useGnVariants, sizeof(bool));
            outStream.write((char *)&useSnVariants, sizeof(bool));
            outStream.write((char *)&fnuScore, sizeof(double));
            outStream.write((char *)&nfnScore, sizeof(double));
            outStream.write((char *)&lnuScore, sizeof(double));
            outStream.write((char *)&nlnScore, sizeof(double));

            outStream.write((char *)&gnSegmentScoreMode,
sizeof(NHSegScoreMode));
            outStream.write((char *)&snSegmentScoreMode,
sizeof(NHSegScoreMode));
            outStream.write((char *)&gnAnchorSegmentMode,
sizeof(NHAnchorSegMode));
            outStream.write((char *)&snAnchorSegmentMode,
sizeof(NHAnchorSegMode));

            outStream.write((char *)&gnAnchorFactor, sizeof(double));
            outStream.write((char *)&snAnchorFactor, sizeof(double));
            outStream.write((char *)&gnOOPSFactor, sizeof(double));
            outStream.write((char *)&snOOPSFactor, sizeof(double));

            outStream.write((char *)&scoreGnTaqs, sizeof(bool));
            outStream.write((char *)&scoreSnTaqs, sizeof(bool));

            outStream.write((char *)&absDelGnTAQFactor, sizeof(double));
            outStream.write((char *)&absDisGnTAQFactor, sizeof(double));
            outStream.write((char *)&absDelSnTAQFactor, sizeof(double));
            outStream.write((char *)&absDisSnTAQFactor, sizeof(double));
            outStream.write((char *)&delGnTAQFactor, sizeof(double));
            outStream.write((char *)&delSnTAQFactor, sizeof(double));
            outStream.write((char *)&disGnTAQFactor, sizeof(double));
            outStream.write((char *)&disSnTAQFactor, sizeof(double));

            outStream.write((char *)&checkGnCompressedName,
sizeof(bool));
            outStream.write((char *)&checkSnCompressedName,
sizeof(bool));

            outStream.write((char *)&gnCompressedNameScore,
sizeof(double));
            outStream.write((char *)&snCompressedNameScore,
sizeof(double));

            outStream.write((char *)&gnScoreThresh, sizeof(double));
            outStream.write((char *)&snScoreThresh, sizeof(double));
            outStream.write((char *)&gnWeight, sizeof(double));
            outStream.write((char *)&snWeight, sizeof(double));

            outStream.write((char *)&gnInitialOnInitialMatchScore,
sizeof(double));
            outStream.write((char *)&snInitialOnInitialMatchScore,
sizeof(double));
      }
      else
            rc = NH_COMP_PARMS_BAD_STREAM_ON_ARCHIVE;

      return rc;
}


NHReturnCode      NHCompParms::setScoreThresh(double aThresh)
{
      NHReturnCode            errorCode;

      if ((aThresh < 0.0) || (aThresh > 1.0))
            errorCode = NH_INVALID_SCORE_THRESH;
      else  {
            scoreThresh = aThresh;
            errorCode = NH_SUCCESS;
      }

      return errorCode;
}


void  NHCompParms::setUseGnLeftBias(bool aBool)
{
      useGnLeftBias = aBool;
}

void  NHCompParms::setUseSnLeftBias(bool aBool)
{
      useSnLeftBias = aBool;
}


void  NHCompParms::setMatchGnIntial(bool aBool)
{
      matchGnIntial = aBool;
}


void  NHCompParms::setMatchSnIntial(bool aBool)
{
      matchSnIntial = aBool;
}


NHReturnCode      NHCompParms::setGnInitialScore(double aScore)
{
      NHReturnCode            errorCode;

      if ((aScore < 0.0) || (aScore > 1.0))
            errorCode = NH_INVALID_GN_INIT_SCORE;
      else  {
```

```
                gnInitialScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setSnInitialScore(double aScore)
{
        NHReturnCode                errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_NH_INIT_SCORE;
        else    {
                snInitialScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setGnInitialOnInitialMatchScore(double
aScore)
{
        NHReturnCode                errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_GN_INIT_ON_INIT_MATCH_SCORE;
        else    {
                gnInitialOnInitialMatchScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setSnInitialOnInitialMatchScore(double
aScore)
{
        NHReturnCode                errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_NH_INIT_ON_INIT_MATCH_SCORE;
        else    {
                snInitialOnInitialMatchScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


void    NHCompParms::setUseGnVariants(bool aBool)
{
        useGnVariants = aBool;
}
```

```
void  NHCompParms::setUseSnVariants(bool aBool)
{
        useSnVariants = aBool;
}


NHReturnCode        NHCompParms::setNFNScore(double aScore)
{
        NHReturnCode              errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_NFN_SCORE;
        else  {
                nfnScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setFNUScore(double aScore)
{
        NHReturnCode              errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_FNU_SCORE;
        else  {
                fnuScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setNLNScore(double aScore)
{
        NHReturnCode              errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_NLN_SCORE;
        else  {
                nlnScore = aScore;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setLNUScore(double aScore)
{
        NHReturnCode              errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_LNU_SCORE;
        else  {
                lnuScore = aScore;
                errorCode = NH_SUCCESS;
        }
        return errorCode;
}

NHReturnCode        NHCompParms::setGnScoreThresh(double aThresh)
{
        NHReturnCode                errorCode;

        if ((aThresh < 0.0) || (aThresh > 1.0))
                errorCode = NH_INVALID_GN_THRESH;
        else  {
                gnScoreThresh = aThresh;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setSnScoreThresh(double aThresh)
{
        NHReturnCode                errorCode;

        if ((aThresh < 0.0) || (aThresh > 1.0))
                errorCode = NH_INVALID_NH_THRESH;
        else  {
                snScoreThresh = aThresh;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setGnWeight(double aWeight)
{
        NHReturnCode                errorCode;

        if ((aWeight < 0.0) || (aWeight > 1.0))
                errorCode = NH_INVALID_GN_WEIGHT;
        else  {
                gnWeight = aWeight;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setSnWeight(double aWeight)
{
        NHReturnCode                errorCode;

        if ((aWeight < 0.0) || (aWeight > 1.0))
                errorCode = NH_INVALID_NH_WEIGHT;
        else  {
                snWeight = aWeight;
                errorCode = NH_SUCCESS;
        }
        return errorCode;
}

void    NHCompParms::setGnSegmentScoreMode(NHSegScoreMode aMode)
{
        gnSegmentScoreMode = aMode;
}

void    NHCompParms::setSnSegmentScoreMode(NHSegScoreMode aMode)
{
        snSegmentScoreMode = aMode;
}

void    NHCompParms::setGnAnchorSegmentMode(NHAnchorSegMode anAnchorMode)
{
        gnAnchorSegmentMode = anAnchorMode;
}

void    NHCompParms::setSnAnchorSegmentMode(NHAnchorSegMode anAnchorMode)
{
        snAnchorSegmentMode = anAnchorMode;
}

NHReturnCode        NHCompParms::setGnAnchorFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_GN_ANCHOR_FACTOR;
        else    {
                gnAnchorFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setSnAnchorFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_NH_ANCHOR_FACTOR;
        else    {
                snAnchorFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setGnOOPSFactor(double aFactor)
{
        NHReturnCode                errorCode;
        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_GN_OOPS_FACTOR;
        else    {
                gnOOPSFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode      NHCompParms::setSnOOPSFactor(double aFactor)
{
        NHReturnCode            errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_NH_OOPS_FACTOR;
        else    {
                snOOPSFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode      NHCompParms::setAbsDelGnTAQFactor(double aFactor)
{
        NHReturnCode            errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_ABS_DEL_GN_TAQ_FACTOR;
        else    {
                absDelGnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode      NHCompParms::setAbsDisGnTAQFactor(double aFactor)
{
        NHReturnCode            errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_ABS_DIS_GN_TAQ_FACTOR;
        else    {
                absDisGnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode      NHCompParms::setAbsDelSnTAQFactor(double aFactor)
{
        NHReturnCode            errorCode;
        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_ABS_DEL_NH_TAQ_FACTOR;
        else    {
                absDelSnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setAbsDisSnTAQFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_ABS_DIS_NH_TAQ_FACTOR;
        else    {
                absDisSnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setDelGnTAQFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_DEL_GN_TAQ_FACTOR;
        else    {
                delGnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


NHReturnCode        NHCompParms::setDelSnTAQFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_DEL_NH_TAQ_FACTOR;
        else    {
                delSnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setDisGnTAQFactor(double aFactor)
{
        NHReturnCode                errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
```

```
                errorCode = NH_INVALID_DIS_GN_TAQ_FACTOR;
        else    {
                disGnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}

NHReturnCode        NHCompParms::setDisSnTAQFactor(double aFactor)
{
        NHReturnCode            errorCode;

        if ((aFactor < 0.0) || (aFactor > 1.0))
                errorCode = NH_INVALID_DIS_NH_TAQ_FACTOR;
        else    {
                disSnTAQFactor = aFactor;
                errorCode = NH_SUCCESS;
        }

        return errorCode;
}


void    NHCompParms::setScoreGnTAQs(bool aBool)
{
        scoreGnTaqs = aBool;
}


void    NHCompParms::setScoreSnTAQs(bool aBool)
{
        scoreSnTaqs = aBool;
}


void            NHCompParms::setCheckGnCompressedName(bool aBool)
{
        checkGnCompressedName = aBool;
}


void            NHCompParms::setCheckSnCompressedName(bool aBool)
{
        checkSnCompressedName = aBool;
}


NHReturnCode                NHCompParms::setGnCompressedNameScore(double
aScore)
{
        NHReturnCode            errorCode;

        if ((aScore < 0.0) || (aScore > 1.0))
                errorCode = NH_INVALID_GN_COMPRESSED_NAME_SCORE;
        else    {
                gnCompressedNameScore = aScore;
                errorCode = NH_SUCCESS;
        }
```

```
      return errorCode;
}


NHReturnCode            NHCompParms::setSnCompressedNameScore(double
aScore)
{
      NHReturnCode            errorCode;

      if ((aScore < 0.0) || (aScore > 1.0))
            errorCode = NH_INVALID_NH_COMPRESSED_NAME_SCORE;
      else  {
            snCompressedNameScore = aScore;
            errorCode = NH_SUCCESS;
      }

      return errorCode;
}


bool        NHCompParms::operator==(NHCompParms &other)
{
      bool  rc;

      rc = ((scoreThresh == other.scoreThresh) &&
                        (useGnLeftBias == other.useGnLeftBias) &&
                        (useSnLeftBias == other.useSnLeftBias) &&
                        (matchGnIntial == other.matchGnIntial) &&
                        (matchSnIntial == other.matchSnIntial) &&
                        (gnInitialScore == other.gnInitialScore) &&
                        (snInitialScore == other.snInitialScore) &&
                        (useGnVariants == other.useGnVariants) &&
                        (useSnVariants == other.useSnVariants) &&
                        (fnuScore == other.fnuScore) &&
                        (nfnScore == other.nfnScore) &&
                        (lnuScore == other.lnuScore) &&
                        (nlnScore == other.nlnScore) &&
                        (gnSegmentScoreMode == other.gnSegmentScoreMode)
&&
                        (snSegmentScoreMode == other.snSegmentScoreMode)
&&
                        (gnAnchorSegmentMode ==
other.gnAnchorSegmentMode) &&
                        (snAnchorSegmentMode ==
other.snAnchorSegmentMode) &&
                        (gnAnchorFactor == other.gnAnchorFactor) &&
                        (snAnchorFactor == other.snAnchorFactor) &&
                        (gnOOPSFactor == other.gnOOPSFactor) &&
                        (snOOPSFactor == other.snOOPSFactor) &&
                        (gnWeight == other.gnWeight) &&
                        (snWeight == other.snWeight) &&
                        (gnScoreThresh == other.gnScoreThresh) &&
                        (snScoreThresh == other.snScoreThresh) &&
                        (scoreGnTaqs == other.scoreGnTaqs) &&
                        (scoreSnTaqs == other.scoreSnTaqs) &&
                        (absDelGnTAQFactor == other.absDelGnTAQFactor)
&&
                        (absDisGnTAQFactor == other.absDisGnTAQFactor)
&&
                        (absDelSnTAQFactor == other.absDelSnTAQFactor)
&&
                        (absDisSnTAQFactor == other.absDisSnTAQFactor)
&&
                        (delGnTAQFactor == other.delGnTAQFactor) &&
                        (delSnTAQFactor == other.delSnTAQFactor) &&
                        (disGnTAQFactor == other.disGnTAQFactor) &&
                        (disSnTAQFactor == other.disSnTAQFactor) &&
                        (checkGnCompressedName ==
other.checkGnCompressedName) &&
                        (checkSnCompressedName ==
other.checkSnCompressedName) &&
                        (gnCompressedNameScore ==
other.gnCompressedNameScore) &&
                        (snCompressedNameScore ==
other.snCompressedNameScore) &&
                        (gnInitialOnInitialMatchScore ==
other.gnInitialOnInitialMatchScore) &&
                        (snInitialOnInitialMatchScore ==
other.snInitialOnInitialMatchScore));
      return rc;
}


NHReturnCode                  NHCompParms::getStatus()
{
      return status;
}
```

```
//    File:  NH_variant_taq_globals.h
//
//    Description:
//
//          Functions to manage the global variant and TAQ resources.
//          We manage the TAQ and variant tables as global resources
//          so that each SNCompParms object does not need to create its
//          own copy of them.  We provide these global functions so that
//          we can control the variables in one location.
//
//
//    History:
//
//          9/08/97     EFB         Created
//          3/20/98     EFB         Changed names to NH from SN
//


#ifndef     NH_VARIANT_TAQ_GLOBALS_DEFFED
#define     NH_VARIANT_TAQ_GLOBALS_DEFFED


#include    "NH_culture_codes.h"

//    function to return pointers to the global SN and GN Variant Tables
NHVariantTable    *NH_getVariantTable(NH_VARIANT_TABLE_TYPES
variantTableType);

NHTAQTable  *NH_getTAQTable();


#endif
```

```
//    File:  NH_variant_taq_globals.cpp
//
//    Description:
//
//          Functions to manage the global variant and TAQ resources.
//          We manage the TAQ and variant tables as global resources
//          so that each NHCompParms object does not need to create its
//          own copy of them.  We provide these global functions so that
//          we can control the variables in one location.
//
//          We should provide some sort of thread protection around
these
//          resources to make sure that two competing threads do not
attempt
//          to grab these resources during creation time.  How can we do
this
//          portably?.
//
//
//    History:
//
//          9/08/97     EFB         Created
//          3/20/98     EFB         Changed names to NH from SN
//


#include    <string.h>


#include    "NH_util.hpp"
#include    "NHVariantTable.hpp"
#include    "NHTAQTable.hpp"

#include    "NH_variant_taq_globals.h"

//    define SN and GN variant tables
NHVariantTable    *NH_snVariantTable = NULL;
NHVariantTable    *NH_gnVariantTable = NULL;

//    define a single TAQ table
NHTAQTable  *NH_taqTable = NULL;



//    functions to create and return pointers to the tables

NHVariantTable    *NH_getVariantTable(NH_VARIANT_TABLE_TYPES
variantTableType)
{
      NHVariantTable    *tablePtr;
      NHVariantTable    **tablePtrPtr = NULL;

      switch      (variantTableType)      {
            case  NH_SURNAME_VARIANTS:
                  tablePtr = NH_snVariantTable;
                  tablePtrPtr = &NH_snVariantTable;
                  break;
            case  NH_GIVENNAME_VARIANTS:
                  tablePtr = NH_gnVariantTable;
                  tablePtrPtr = &NH_gnVariantTable;
                  break;
            default:
                  tablePtr = NULL;
      }
      if (tablePtr == NULL)    {
            tablePtr = new
NHVariantTable(variantTableType);         //    create the table
            if (tablePtrPtr != NULL)
                  *tablePtrPtr = tablePtr;      //    assign the global
variable
      }
      return      tablePtr;
}


NHTAQTable  *NH_getTAQTable()
{
      if (NH_taqTable == NULL)      {
            NH_taqTable = new
NHTAQTable(NH_PRODUCTION_TAQ_TABLE);          //    create the table
      }
      return      NH_taqTable;
}
```

```
//    File:  NH_util.cpp
//
//    Description:
//
//          Implementation of various utility functions used in the
SNAPI
//
//
//    History:
//
//          5/15/97     EFB         Created
//          3/20/98     EFB         Changed names to NH from SN
//


#include    <string.h>


#include    "NH_util.hpp"
#include    "NHCompParms.hpp"



//    function to remove leading and trailing spaces from a string
//    in place.
//  Strips the string at either end or both ends.
//  Stripchars specify the characters that should
//  be stripped.  We start by seeing if they want the
//  trailing chars stripped, which is easy.  We simply
//  work backwards from the end of the string, looking for
//  the first non-strippable character, and terminate the
//  string just past that character.  Then if they wanted
//  leading chars stripped, we work forwards to the first
//  non-strippable char, and then move that and each following
//  char to the beginning of the string.
void  NH_strip(char *aString)
{
  char *end_point;
  char *ch;
  int  len;

  if ((len = strlen(aString)) != 0) { //  if there is a string
             //  start at end
            end_point = aString + len - 1;

             //  and work back till we get a non-space or get to
             //  the begining of our string, chopping off what's left.
             //  Also make sure we don't zoom right past the beginning of
the
             //  string.
            for (; strchr(NH_DEFAULT_WHITESPACE, *end_point) != NULL &&
end_point != aString; end_point--)
                  ;
             //  if string was all whitespace
            if ((end_point == aString) && strchr(NH_DEFAULT_WHITESPACE,
*aString) != NULL)
                  *aString = EOS;   //  erase it all, and we're done,
could return here
            else
                  *(end_point + 1) = EOS;    //  just chop off excess
blanks

            //  make sure there is still a string, since it might
            //  have been stripped entirely above.
            if (*aString) {
                  //  now find first non space.  we know string has at
least one
                  //  nonwhite space, so we don't have to check for
NULL.
                  for (ch = aString; strchr(NH_DEFAULT_WHITESPACE, *ch)
!= NULL; ch++)
                        ;
                  if (ch != aString) { // if there were leading spaces,
move the block back
                        char  *target = aString;
                        while (*ch != EOS) {
                              *target = *ch;
                              target++;
                              ch++;
                        }
                        //  and get the null char also
                        *target = *ch;
                  } //  end if (are there leading spaces?)
            } //  end if (and text left?)
  } //  end (is there a string at all ?)
}


char  *     NH_strrchr(char *stringStart, char *searchPos, char
searchChar)
{
      while (1)   {
            if (*searchPos == searchChar)
                  break;
            if (searchPos == stringStart) {
                  searchPos = NULL;       //    string not found, so
return NULL
                  break;
            }
            searchPos--;
      }
      return searchPos;
}
```

```
//
//    File:  NH_queens_arrays.hpp
//
//    Description:
//
//          Contains global definitions and declarations for the valid
//          combinations of indexes for the best score calculation
//
//
//    History:
//
//          6/4/97      EFB         Created
//          3/20/98     EFB         Changed names to NH from SN
//

typedef     unsigned char byte;

      byte  twoByTwo[] = {1, 0,
                                                    0, 1};

      byte  twoByThree[] = {  1, 2,

1, 0,

2, 1,

2, 0,

0, 1,

0, 2};
      byte  twoByFour[] = {   1, 2,

1, 3,

1, 0,

2, 1,

2, 3,

2, 0,

3, 1,

3, 2,

3, 0,

0, 1,

0, 2,

0, 3};
      byte  twoByFive[] = {   1, 2,

1, 3,
1, 4,
1, 0,
2, 1,
2, 3,
2, 4,
2, 0,
3, 1,
3, 2,
3, 4,
3, 0,
4, 1,
4, 2,
4, 3,
4, 0,
0, 1,
0, 2,
0, 3,
0, 4};
      byte  threeByThree[] = {      1, 2, 0,
      1, 0, 2,
      2, 1, 0,
      2, 0, 1,
      0, 1, 2,
      0, 2, 1};
      byte  threeByFour[] = { 1, 2, 3,
      1, 2, 0,
      1, 3, 2,
      1, 3, 0,
      1, 0, 2,
      1, 0, 3,
2, 1, 3,
2, 1, 0,
2, 3, 1,
2, 3, 0,
2, 0, 1,
2, 0, 3,
3, 1, 2,
3, 1, 0,
3, 2, 1,
3, 2, 0,
3, 0, 1,
3, 0, 2,
0, 1, 2,
0, 1, 3,
0, 2, 1,
0, 2, 3,
0, 3, 1,
0, 3, 2};
byte  threeByFive[] = { 1, 2, 3,
1, 2, 4,
1, 2, 0,
1, 3, 2,
1, 3, 4,
1, 3, 0,
1, 4, 2,
1, 4, 3,
1, 4, 0,
1, 0, 2,
1, 0, 3,
1, 0, 4,
2, 1, 3,
2, 1, 4,
2, 1, 0,
2, 3, 1,
2, 3, 4,
2, 3, 0,
2, 4, 1,
2, 4, 3,
2, 4, 0,
2, 0, 1,
2, 0, 3,
2, 0, 4,
3, 1, 2,
3, 1, 4,
3, 1, 0,
3, 2, 1,
3, 2, 4,
3, 2, 0,
3, 4, 1,
3, 4, 2,
3, 4, 0,
3, 0, 1,
3, 0, 2,
3, 0, 4,
4, 1, 2,
4, 1, 3,
4, 1, 0,
4, 2, 1,
4, 2, 3,
4, 2, 0,
4, 3, 1,
4, 3, 2,
4, 3, 0,
4, 0, 1,
4, 0, 2,
4, 0, 3,
0, 1, 2,
0, 1, 3,
0, 1, 4,
0, 2, 1,
0, 2, 3,
0, 2, 4,
0, 3, 1,
0, 3, 2,
0, 3, 4,
0, 4, 1,
0, 4, 2,
0, 4, 3};
byte  fourByFour[] =    {     1, 2, 3, 0,
1, 2, 0, 3,
1, 3, 0, 2,
1, 3, 2, 0,
1, 0, 2, 3,
1, 0, 3, 2,
2, 1, 3, 0,
2, 1, 0, 3,
2, 3, 1, 0,
2, 3, 0, 1,
2, 0, 1, 3,
2, 0, 3, 1,
3, 1, 2, 0,
3, 1, 0, 2,
3, 2, 1, 0,
3, 2, 0, 1,
3, 0, 1, 2,
3, 0, 2, 1,
0, 1, 2, 3,
0, 1, 3, 2,
0, 2, 1, 3,
0, 2, 3, 1,
0, 3, 1, 2,
0, 3, 2, 1};
byte  fourByFive[] =    {     1, 2, 3, 4,
1, 2, 3, 0,
1, 2, 4, 3,
1, 2, 4, 0,
1, 2, 0, 3,
1, 2, 0, 4,
1, 3, 2, 4,
1, 3, 2, 0,
1, 3, 4, 2,
1, 3, 4, 0,
1, 3, 0, 2,
1, 3, 0, 4,
1, 4, 2, 3,
1, 4, 2, 0,
1, 4, 3, 2,
1, 4, 3, 0,
1, 4, 0, 2,
1, 4, 0, 3,
1, 0, 2, 3,
1, 0, 2, 4,
1, 0, 3, 2,
1, 0, 3, 4,
1, 0, 4, 2,
1, 0, 4, 3,
2, 1, 3, 4,
2, 1, 3, 0,
2, 1, 4, 3,
2, 1, 4, 0,
2, 1, 0, 3,
2, 1, 0, 4,
2, 3, 1, 4,
2, 3, 1, 0,
2, 3, 4, 1,
2, 3, 4, 0,
2, 3, 0, 1,
2, 3, 0, 4,
2, 4, 1, 3,
2, 4, 1, 0,
2, 4, 3, 1,
2, 4, 3, 0,
2, 4, 0, 1,
2, 4, 0, 3,
2, 0, 1, 3,
2, 0, 1, 4,
2, 0, 3, 1,
2, 0, 3, 4,
2, 0, 4, 1,
2, 0, 4, 3,
3, 2, 1, 4,
3, 2, 1, 0,
3, 2, 4, 1,
3, 2, 4, 0,
3, 2, 0, 1,
3, 2, 0, 4,
3, 1, 2, 4,
3, 1, 2, 0,
3, 1, 4, 2,
3, 1, 4, 0,
3, 1, 0, 2,
3, 1, 0, 4,
3, 4, 2, 1,
3, 4, 2, 0,
3, 4, 1, 2,
3, 4, 1, 0,
3, 4, 0, 2,
3, 4, 0, 1,
3, 0, 2, 1,
3, 0, 2, 4,
3, 0, 1, 2,
3, 0, 1, 4,
3, 0, 4, 2,
3, 0, 4, 1,
4, 2, 3, 1,
4, 2, 3, 0,
4, 2, 1, 3,
4, 2, 1, 0,
4, 2, 0, 3,
4, 2, 0, 1,
4, 3, 2, 1,
4, 3, 2, 0,
4, 3, 1, 2,
/* Generated by VariantManager */
addVariant("ANN","ANITA",0.85,"E ");
addVariant("ANN","ANA",0.85,"E ");
addVariant("ANN","ANNIE",0.90,"E ");
addVariant("ANN","ANNA",0:85,"E ");
addVariant("ANN","ANNE",0.95,"E ");
addVariant("ANN","ANNETTE",0.85,"E ");
```

```
/* Generated by VariantManager */
addVariant("SON","SWUN",0.95,"C ");
addVariant("SON","SHON",0.95,"K ");
addVariant("SON","SOHN",0.95,"K ");
```